6.867 Notes on Graphical models

Contents

1 Intro

We are now going to focus on the problem of *density estimation*. In density estimation, we are given a set of data $\mathcal{D} = \{x^{(1)}, \dots, x^{(n)}\}$ that we assume are drawn IID from some distribution $\Pr(X)$. Our job is to estimate that distribution.

We have seen versions of this problem already twice before:

- When building a probabilistic discriminative model for regression or classification, we estimate $\Pr(Y \mid X; \theta)$. So, for a given x value, we are delivering an estimate of the distribution on Y; however, in the case of discriminative learning, the whole conditional distribution is parameterized by θ, so we are not typically concentrating directly on representing a density on Y.

- When building a probabilistic generative model for regression or classification, we convert the problem of estimating $\Pr(Y \mid X)$ to one of estimating $\Pr(Y, X; \theta)$, knowing that $\Pr(Y \mid X = x) = \Pr(Y, X = x)/\Pr(X = x)$. That is a density estimation problem, on the joint distribution of Y and X. In the case of classification, we further convert this to a problem of estimating $\Pr(Y; \theta_1)$ and $\Pr(X \mid Y; \theta_2)$.

Other times, we may be interested in density estimation as our primary problem. It can be useful for:

- Building models that reveal something about the underlying structure of a collection of objects

- Doing *anomaly detection*; that is, after training on $\mathcal{D}$, given a new x, predict its likelihood in the distribution the training data was drawn from. This may let us discover unlikely or unusual events.

But we tend to find that *unlikely* doesn't usually mean *interesting*.

	No model	Prediction rule	Prob model	Dist over models
Density estimation			❃	

The distinction between a prediction rule and a probabilistic model doesn't make much sense here, since we're predicting probabilities. Later on, we will look at non-parametric methods, which can also be seen as doing density estimation, but with a model that is based directly on the training data.

We also use graphical models, as researchers and practitioners, as a way of describing the models we are applying to solve a problem.

2 Representation and semantics

Barber's book is a gentler introduction than Bishop, if you want something with more examples: `http://web4.cs.ucl.ac.uk/staff/D.Barber/textbook/270212.pdf` .

The idea behind graphical models is that we can decompose the problems of estimating a joint distribution and of reasoning with it (for instance, computing $\Pr(Y \mid X = x)$), into a set of related but easier sub-problems, based on

- Factoring the domain into multiple random variables $X_1, \dots, X_d$; the random variables may be discrete or continuous

- Describing $\Pr(X_1, \dots, X_d)$ as a product of factors $\prod_k \phi_k(X)$, where each factor ϕ_k depends only on a subset of the random variables X_i.

By factoring the distribution in this way, we improve the computational efficiency of reasoning in the model and decrease the number of samples required for learning.

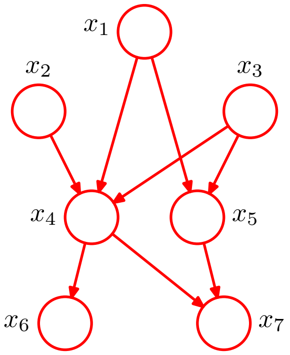


Figure 1: Bayes net (Bishop Figure 8.2)

These models are called *graphical models* because we use graphs to model the dependency structure among the variables that is induced by the factors ϕ.

In addition, it is often the case that a good factoring of the domain can be provided based on human insight into the structure of the problem; graphical models let us *easily combine structural prior knowledge with statistical data*.

Graphical models are *multi-purpose*: if we learn a (compact) joint distribution on $X_1, \ldots, X_d$, then we can compute any conditional distribution $\Pr(X_{i_1,\ldots,i_k} \mid X_{j_1,\ldots j_m})$. We can condition on any evidence available, and ask about any marginal or joint distribution over the other variables.

2.1 Directed models: Bayesian networks

We will first consider *directed* graphical models, also known as *Bayesian networks*. They are made up of:

- **Nodes**: representing random variables
- **Arcs**: representing dependencies between variables

We will tend to focus on the discrete case, but the continuous case is interesting and important. Some combinations of discrete and continuous nodes can be challenging to deal with.

Together, the arcs and nodes must constitute a *directed acyclic graph* (DAG). This means that there must be no *directed* cycles among the arcs. The graph in figure 1 is a DAG: although there are cycles, none of them are directed cycles (you can't follow it all the way around in the direction of the arrows).

In figure 1,

- x_1 and x_2 are *parents* of x_4
- x_1 is an *ancestor* of x_6
- x_6 is a *descendant* of x_1
- Any two nodes that share a parent are *siblings*

Each node X_i contains a definition for the conditional distribution

$$\Pr(X_i \mid \text{Parents}(X_i)) \quad .$$

Generally, it takes many fewer parameters to specify a Bayes net than to specify the whole joint distribution.

The joint distribution is a product of the individual factors. This is sometimes known as the *chain rule* of Bayesian networks:

$$\Pr(X) = \prod_i \Pr(X_i \mid \text{Parents}(X_i)) \ \ .$$

So,

$$\begin{aligned}
\Pr(X_1 = x_1, \ldots, X_7 = x_7) &= \Pr(X_1 = x_1)\Pr(X_2 = x_2)\Pr(X_3 = x_3) \\
&\quad \Pr(X_4 = x_4 \mid X_1 = x_1, X_2 = x_2, X_3 = x_3) \\
&\quad \Pr(X_5 = x_5 \mid X_1 = x_1, X_3 = x_3)\Pr(X_6 = x_6 \mid X_4 = x_4) \\
&\quad \Pr(X_7 = x_7 \mid X_4 = x_4, X_5 = x_5)
\end{aligned}$$

Semantics of arcs

First, we define conditional independence: A is conditionally independent of B given C, written $A \perp B \mid C$, iff

$$\Pr(A \mid B, C) = \Pr(A \mid C) \ \ .$$

I think about this as saying "If I know the value of C, then knowing B doesn't give me any further information about A."

Bayesian networks make the *local Markov assumption* (LMA): the random variable represented by a node X_i in the graph is *conditionally independent* of its *non-descendants* given its *parents*. That is, for all X_i,

$$X_i \perp \text{Nondescendants}(X_i) \mid \text{Parents}(X_i) \ \ .$$

It is the absence of links that encodes independence information.

These structures make the same conditional independence assertions:

```
A  -> B  -> C
A <- B <- C
A <- B -> C
```

The LMA in the first structure asserts C is conditionally independent of A given B:

$$\Pr(C \mid A, B) = \Pr(C \mid B) \ \ .$$

The LMA in the second structure asserts A is conditionally independent of C given B:

$$\Pr(A \mid C, B) = \Pr(A \mid B) \ \ .$$

The LMA in the third structure asserts both:

$$\begin{aligned}
\Pr(C \mid A, B) &= \Pr(C \mid B) \\
\Pr(A \mid C, B) &= \Pr(A \mid B)
\end{aligned}$$

To show these are all equivalent, we really only need to show that the first implies the second:

$$\begin{aligned}
\Pr(C \mid A, B) &= \Pr(C \mid B) \\
\frac{\Pr(A, B, C)}{\Pr(A, B)} &= \frac{\Pr(B, C)}{\Pr(B)} \\
\Pr(A, B, C) &= \frac{\Pr(A, B)\Pr(B, C)}{\Pr(B)} \\
\frac{\Pr(A, B, C)}{\Pr(B, C)} &= \frac{\Pr(A, B)}{\Pr(B)} \\
\Pr(A \mid B, C) &= \Pr(A \mid B)
\end{aligned}$$

This structure is different

```
A -> B <- C
```

The LMA doesn't make any conditional independence assertions in this case.

To further explore this, let's consider the number of parameters in these models. Assuming each variable is binary, then in the top model, we need to specify $\Pr(A)$, $\Pr(B \mid A)$ and $\Pr(C \mid B)$. To specify $\Pr(A)$, we just need one parameter: $\Pr(A = 1)$. Each of the other two *conditional probability tables* (CPTs) requires 2 parameters:e.g., $\Pr(B = 1 \mid A = 1)$ and $\Pr(B = 1 \mid A = 0)$. So, to specify the whole network, in each of the top three cases, we need 5 parameters.

To specify this last network, we need to specify $\Pr(A)$, $\Pr(C)$, and $\Pr(B \mid A, C)$. It takes one parameter for each of the first two distributions, and then 4 for the last one. So, this is a more expressive model.

Exercise: How many parameters does it take to specify $\Pr(A, B, C)$ with structural assumptions at all?

Exercise: Give a probability distribution that (1) cannot be expressed in the top three models; (2) can be expressed in the bottom model.

Example: explaining away

Consider the network

```
Battery -> Gauge <- FuelTank
```

Here are some CPTs:

$$\begin{aligned}
\Pr(B = 1) &= 0.9 \\
\Pr(F = 1) &= 0.9 \\
\Pr(G = 1 \mid B = 1, F = 1) &= 0.8 \\
\Pr(G = 1 \mid B = 1, F = 0) &= 0.2 \\
\Pr(G = 1 \mid B = 0, F = 1) &= 0.2 \\
\Pr(G = 1 \mid B = 0, F = 0) &= 0.1
\end{aligned}$$

- What is the prior that the tank is empty?

$$\Pr(F = 0) = 0.1$$

- What if we observe the fuel gauge and find that it reads empty?

$$\begin{aligned}
\Pr(F = 0 \mid G = 0) &= \frac{\Pr(G = 0 \mid F = 0)\Pr(F = 0)}{\Pr(G = 0)} \\
&= \frac{\sum_b \Pr(G = 0, B = b \mid F = 0)\Pr(F = 0)}{\sum_{f,b} \Pr(G = 0, F = f, B = b)} \\
&= \frac{\sum_b \Pr(G = 0 \mid F = 0, B = b)\Pr(B = b \mid F = 0)\Pr(F = 0)}{\sum_{f,b} \Pr(G = 0 \mid F = f, B = b)\Pr(F = f, B = b)} \\
&= \frac{\sum_b \Pr(G = 0 \mid F = 0, B = b)\Pr(B = b)\Pr(F = 0)}{\sum_{f,b} \Pr(G = 0 \mid F = f, B = b)\Pr(F = f)\Pr(B = b)} \\
&= \frac{(0.8 \cdot 0.9 + 0.9 \cdot 0.1) \cdot 0.1}{(0.8 \cdot 0.9 + 0.9 \cdot 0.1) \cdot 0.1 + (0.2 \cdot 0.9 + 0.8 \cdot 0.1) \cdot 0.9} \\
&\approx 0.257
\end{aligned}$$

This is greater than $\Pr(F = 0)$; so, seeing the gauge reading empty makes us think it is more likely that the fuel tank is empty. Seems reasonable.

- Now, what if we find the battery is dead?

$$\begin{aligned}
\Pr(F=0 \mid G=0, B=0) &= \frac{\Pr(G=0 \mid F=0, B=0)\Pr(F=0 \mid B=0)}{\Pr(G=0 \mid B=0)} \\
&= \frac{\Pr(G=0 \mid F=0, B=0)\Pr(F=0)}{\sum_f \Pr(G=0, F=f \mid B=0)} \\
&= \frac{\Pr(G=0 \mid F=0, B=0)\Pr(F=0)}{\sum_f \Pr(G=0 \mid F=f, B=0)\Pr(F=f \mid B=0)} \\
&= \frac{\Pr(G=0 \mid F=0, B=0)\Pr(F=0)}{\sum_f \Pr(G=0 \mid F=f, B=0)\Pr(F=f)} \\
&= \frac{0.9 \cdot 0.1}{0.9 \cdot 0.1 + 0.8 \cdot 0.9} \\
&= 0.111
\end{aligned}$$

  The probability that the tank is empty has *decreased*! Finding that the battery is flat *explains away* the empty fuel tank reading.

Conditional independence

Node A is *d-separated* from node B given set of nodes C iff all paths from A to B are blocked by C. A path is *blocked* if and only if it includes a node such that either:

- The arrows on the path meet either head-to-tail or tail-to-tail at the node, and the node is in C; or
- The arrows meet head-to-head at the node, and neither the node nor any of its descendants, is in C.

Theorem: If A is d-separated from B given C, then A is conditionally independent of B given C.

A useful concept is the *Markov blanket* of a node X_i. It consists of:

- The parents of X_i
- The children of X_i
- The parents of the children of X_i

Theorem: X_i is conditionally independent of all other nodes in the graph, Given the values of the nodes in the Markov blanket of X_i.

2.2 Undirected models: Markov random fields

A Markov random field has

- **Nodes** representing random variables
- **Arcs** representing the existence of dependencies between variables
- **Potential functions** representing the numerical dependencies between groups of variables; generally one potential per *clique* in the graph, but okay to have one potential per *maximum clique*

In undirected models, the Markov blanket of a node is simply its set of neighbors.

The equivalent notion to d-separation is simply separation in the graph: A is separated from B by the set C if, in the graph with the nodes in C removed, there are no paths from A to B.

The joint probability distribution is

$$\Pr(X_1, \ldots, X_d) = \frac{1}{Z} \prod_C \varphi_C(X_c) \ ,$$

where C ranges over all of the cliques in the graph, φ_C is a function from assignments of values to the variables in clique C to positive values (*they do not need to be probabilities!!*), and Z is a normalizing constant which is called the *partition function*:

$$Z = \sum_x \prod_C \varphi_C(x_c) \ .$$

Hammersley-Clifford theorem: The set of probability distributions with conditional independence relations consistent with the graph structure of an MRF is the same as the set of probability distributions describable as a normalized product of potentials over the maximal cliques of the graph.

- Good: potentials don't have to be normalized; easy to describe a model
- Bad: hard to compute Z.
- Bad: hard to estimate φ from data.

Bishop has a nice example application of image de-noising: one random variable for each observed pixel; one for each actual pixel. Potentials are such that the actual pixel value strongly wants to have the same value as the observed value of that pixel; and that neighboring pixels would prefer to have same or similar values. The opinions of several agreeing neighbors can "override" a noisy observation value.

2.3 Directed and undirected graphs

Given a directed graph, it is very easy to make an undirected graph.

- Add edges to the undirected graph so that it contains a clique for each set $\{X_i\} \cup \text{Parents}(X_i)$. This process is called *moralization*.
- Add a potential function

$$\varphi_i(\{X_i\} \cup \text{Parents}(X_i)) = \Pr(X_i \mid \text{Parents}(X_i))$$

Somewhat annoyingly, but yet memorably, because what we have to do is marry the parents of X_i.

In general, this will result in a model with more, smaller, clique potentials than necessary. It will always be possible to convert this into a model with potentials only on the maximal cliques.

It is important to note that, in the process of going from a directed model to an undirected model, we generally lose the ability to express, graphically, all of the conditional independence relations we had in the original model.

They are, of course, still there in the distribution, but they may be harder to "see".

Given an undirected graph, it is generally much trickier to make a directed graph. It's not often done and we won't address it here.

It is also possible to make graphs that contain both directed and undirected edges, but that is also a story for another time.

2.4 Factor graphs

A really useful unifying representation for directed and undirected graphs is something called a *factor graph*.

Factor graphs have two kinds of nodes, and are *bipartite*:

- A node representing a random variable is connected to each of the nodes representing factors to which it is an argument;
- A node representing a factor is connected to each of the random variables that is an argument to that factor.

Factor graphs make more representation details clear.

To convert an undirected graph to a factor graph: make a variable node for each variable node in the undirected graph; make a factor node for each maximal clique in the graph; connect each variable node to the factor nodes for any cliques it is a member of.

To convert a directed graph to a factor graph: make a variable node for each variable node in the directed graph; make a factor node for each CPT in the directed graph; connect every variable mentioned in the CPT for a factor node to that factor node.

Bishop's figures 8.41 and 8.42 give good examples of factor graphs.

Trees are easy! If we start with a directed or indirected tree, then the corresponding factor graph will be a tree. If we start with a *directed polytree* (which is a DAG with no cycles at all, directed or undirected), then although the corresponding undirected model has cycles in it, the associated factor graph will not.

We'll see what this means in a little bit...for now, just remember that we like trees.

If, however, we start with an undirected model that has loops or with a directed model that is a DAG but not a tree or polytree, then the resulting factor graph will not be a tree. Sadface.

3 Exact inference

There are two inference problems that we commonly find useful to solve:

- *Conditional probability*: $\Pr(Y \mid E = e)$, where Y and E are sets of variables, $Y \subset X$, $E \subset X$, and generally $Y \cap E = \emptyset$; and e is the observed values of the variables in E. Note that it is not necessary that $Y \cup E = X$: generally, there will be some additional variables that are neither observed nor in the query, which will need to be marginalized out.
- *Most probable assignment (MAP)*:

$$\arg\max_y \Pr(Y = y \mid E = e) \ .$$

Note that the MAP of a set of variables is not necessarily the set of MAPs of the individual variables.

A conditional probability query in which $E = \emptyset$ corresponds to asking for a marginal distribution on one or more variables.

We will find that: inference on factor graphs that are trees is polynomial in the sizes of the factors. Inference on non-trees is generally exponential in a measure called the "tree width". Generally, we can exploit the graphical structure of a model to make inference efficient.

We'll focus on conditional probability queries. Letting $Z = X \setminus (Y \cup E)$, we can write

Just to keep things simple-looking, we are using y to range over all assignments of values to the Y variables; we'll use e and z similarly.

$$\begin{aligned}\Pr(Y = y \mid E = e) &= \frac{\Pr(Y = y, E = e)}{\Pr(E = e)} \\ &= \frac{\sum_z \Pr(Y = y, Z = z, E = e)}{\sum_{y,z} \Pr(Y = y, Z = z, E = e)}\end{aligned}$$

So, if our graphical model describes a joint probability distribution on $\Pr(X) = \Pr(Y, Z, E)$, then all we have to be able to do is compute two different marginals and divide.

This is easier said than done, though: the number of possible assignments we will have to add up is exponential in the number of unobserved variables.

3.1 Variable elimination

3.1.1 Simple chain

Let's start with a simple case, in which the nodes are connected in a chain. Converting from directed to undirected is trivial. The corresponding factor graph just has a factor between each pair of nodes. The models are shown in Bishop figure 8.32

The joint distribution is:

$$\Pr(x) = \frac{1}{z}\phi_{1,2}(x_1, x_2)\phi_{2,3}(x_2, x_3)\cdots\phi_{n-1,n}(x_{n-1}, x_n) \quad .$$

Can we be clever about marginalizing? Let's look at a simple chain with three nodes, where we want to marginalize out the nodes on each end:

We don't need a normalizing constant z here, because these potentials are the CPDs from a directed network and are, hence, already normalized.

$$\begin{aligned}\Pr(X_2 = x_2) &= \sum_{x_1}\sum_{x_3}\phi_{1,2}(x_1, x_2)\phi_{2,3}(x_2, x_3)\\ &= \sum_{x_1}\phi(x_1, x_2)\cdot\sum_{x_3}\phi_{2,3}(x_2, x_3)\\ &= \mu_\alpha(x_2)\cdot\mu_\beta(x_2)\end{aligned}$$

where $\mu_\alpha(X_j)$ is a function mapping values of a set of variables X_j to $\mathbb{R}^+$.It is called μ for *message*. Here we have a *forward message*, μ_α and a *backward message*, μ_β. Because there are a finite number of values of X_j, you can also think of it as a vector of values: then the dot product operation between them is the usual vector dot product.

Throughout this section, we will be computing new factors or a special kind of factor called a message: you can think of these as: functions, from values of a set of random variables to real numbers, indicating how "compatible" those value are with one another; or as tables representing those functions; or as unnormalized probability distributions. In the case of messages, they will be indexed by a single variable (or, in the function view, have a single argument). In the case of factors, they may be indexed by multiple variables.

What's good about this? We were able to do two independent sums, one over values for X_1 and one over values for X_3, instead of a quadratic operation. This is a (very simple) instance of a more general algorithm called variable elimination. Returning to the general case of the chain, we can compute a marginal over a variable in the middle with:

$$\begin{aligned}\Pr(X_k = x_k) &= \frac{1}{z}\left(\sum_{x_{k-1}}\phi_{k-1,k}(x_{k-1}, x_k)\cdots\left(\sum_{x_2}\phi_{2,3}(x_2, x_3)\left(\sum_{x_1}\phi_{1,2}(x_1, x_2)\right)\right)\cdots\right)\cdot\\ &\quad\left(\sum_{x_{k+1}}\phi_{k,k+1}(x_k, x_{k+1})\cdots\left(\sum_{n}\phi_{n-1,n}(x_{n-1}, x_n)\right)\cdots\right)\\ &= \mu_\alpha(x_k)\cdot\mu_\beta(x_k)\end{aligned}$$

This takes $O(nm^2)$ for n nodes with m values each; much less than $O(m^n)$!!

The fundamental properties we are taking advantage of, here, is the distribution of summation over a product: $ab + ac = a(b + c)$. When we do that, we save an arithmetic operation.

3.1.2 More generally

To compute a single marginal distribution, we can use the *variable elimination* algorithm. It can be applied to any graphical model, whether or not it has directed or undirected loops. We will just present it informally.
To compute $\Pr(X_j)$

- $V = \{X_1, \ldots, X_{j-1}, X_j, \ldots, X_n\}$ is a set of variables to be eliminated
- $F = \{\phi_1, \ldots, \phi_m\}$ is a set of factors that are still relevant to the problem, initialized to contain all the factors in the model
- Loop until $|F| = 1$:

  - Select a variable $W \in V$ to eliminate
  - Find all factors $\Phi \subseteq F$ that take W as an argument
  - Let Z be the set of variables involved in some factor in Φ, but not including W
  - Construct a new factor, ϕ_{new} with arguments Z, where

$$\phi_{new}(z) = \sum_{w} \prod_{\phi \in \Phi} \phi(w, z) \quad .$$

  - $V = V \setminus \{W\}$
  - $F = F \cup \{\phi_{new}\} \setminus \Phi$

At termination F contains a single factor on X_j. If the original potentials were unnormalized, then it will be necessary to normalize it in order to get $\Pr(X_j$.

Variable elimination can be applied to any directed model (it need not be a tree).

- It will take time exponential in the number of variables in the largest factor ϕ_{new} that gets created.
- The size of the largest factor depends on the order in which we eliminate variables.
- It is NP-complete to find the best variable ordering. But there are some reasonable heuristics.
- The number of variables in the largest factor using the best ordering is called the *tree width*.
- This problem cannot be solved faster than exponentially in the tree width no matter what algorithm we use.

3.2 Message passing

We will explore an alternative inference algorithm, called *message passing* or *belief propagation*. It really only applies to models (directed or undirected) whose factor graphs are trees.

It is useful because:

- It can be used to compute all the marginals in the same amount of time it takes variable elimination.
- It can be used to compute a MAP assignment as well as a marginal or conditional probability.
- It can actually be applied to factor graphs that aren't trees...but the results are not exactly correct in that case.

3.2.1 On a chain

We can compute the necessary quantities on the chain recursively, thinking of messages μ_α as propagating from left to right in the graph and messages μ_β as propagating from right ot left. Whenever we know both α and β messages for a node, we can compute its distribution.

Remember that a message μ_{x_k} is a vector of m_k values, where m is the number of values that x_k can take on. Now, the recursive definitions, first for the *forward messages*:

Think of these as expressing a collective opinion, based on all the nodes with indices lower than k, about the values of x_k.

$$\begin{aligned}\mu_\alpha(x_k) &= \sum_{x_{k-1}} \phi_{k-1,k}(x_{k-1}, x_k)\left(\sum_{x_{k-2}} \cdots\right)\\ &= \sum_{x_{k-1}} \phi_{k-1,k}(x_{k-1}, x_k)\cdot \mu_\alpha(x_{k-1})\end{aligned}$$

The base case is

$$\mu_\alpha(x_1) = \mathbf{1}\ \ .$$

Now, for the *backward messages*:

Think of these as expressing a collective opinion, based on all the nodes with indices higher than k, about the values of x_k.

$$\begin{aligned}\mu_\beta(x_k) &= \sum_{x_{k+1}} \phi_{k,k+1}(x_k, x_{k+1})\left(\sum_{x_{k+2}} \cdots\right)\\ &= \sum_{x_{k+1}} \phi_{k,k+1}(x_k, x_{k+1})\cdot \mu_\beta(x_{k+1})\end{aligned}$$

The base case is

$$\mu_\beta(x_n) = \mathbf{1}\ \ .$$

One pass of "message passing" in each direction along the whole chain yields *all marginal distributions*.

If we have *observed* variable X_j to have value v_j, then we

- Add one more potential
$$\phi_{\text{obs}}(x_j) = \begin{cases}1 & \text{if } x_j = v_j\\ 0 & \text{otherwise}\end{cases}$$
- Multiply it into $\phi_{j-1,j}$ and $\phi_{j,j+1}$

3.2.2 On a factor graph that is tree

Now we'll generalize the algorithm to apply to factor graphs. Think of the variable X_k. It is connected to a set of factors, which we will call neighbors(X_k). On the "far side" of each factor is a whole set of other factors. These factor sets do not overlap. We will express the marginal at node X_k as a product of new factors F_s, one for each neighbor ϕ_s of X_k, each of which summarizes the effects of the whole set of variables X_w and factors that are on the far side of factor ϕ_s.

We can express the marginal at node k as:

$$\begin{aligned}\Pr(x_k) &= \sum_{x-\{x_k\}} \Pr(x)\\ &= \sum_{x-\{x_k\}} \prod_i \phi_i(x)\\ &= \sum_{x-\{x_k\}} \prod_{S\in\text{neighbors}(X_k)} F_s(x_k, z_s)\end{aligned}$$

where z_S is the set of variables that are in the tree rooted at factor S, but not including the children of X_k and $F_s(x_k, z_s)$ represents the product of all the factors in that tree.

Exchanging sums and products in a similar way as for the chain, and adding in one term for each factor S connected to X_k, we can write this as a product of *incoming messages*:

$$\begin{aligned}\Pr(x_k) &= \prod_{S\in\text{neighbors}(X_k)} \sum_{x_s} F_s(x_k, x_s)\\ &= \prod_{S\in\text{neighbors}(X_k)} \mu_{\phi_s\to X_k}(x_k)\end{aligned}$$

Now we can express a recursive algorithm in which messages are passed “inward,” from the leaves of the tree. There are two kinds of messages: those going from factors to nodes, and those going from nodes to factors.

This is a pretty abstract description. There is a small worked example in the recitation handout.

Factor-to-node messages: Let X_S be the set of variables connected to factor ϕ_S. To compute a message from ϕ_S to X_k, we take the sum over all the values of variables in ϕ_S except for X_k, of the product of the factor ϕ_s applied to those values times the product, over all the all the variables X_m that are neighbors of ϕ_k except for X_k, of the messages going from those variables into ϕ_S:

$$\mu_{\phi_S \to X_k}(x_k) = \sum_{x_S \setminus x_k} \phi_s(x_s) \prod_{X_m \in X_s \setminus X_k} \mu_{X_m \to \phi_S}(x_m) \ .$$

Base case if ϕ is a leaf:

$$\mu_{\phi \to X_i}(x_i) = \phi(x_i) \ .$$

Node-to-factor messages: There is a similar method for computing messages from nodes to factors:

$$\mu_{X_m \to \phi_S}(x_m) = \prod_{\phi_l \in \text{neighbors}(X_m) \setminus \phi_S} \mu_{\phi_l \to X_m}(x_m) \ .$$

Base case if X_i is a leaf:

$$\mu_{X_i \to \phi}(x_i) = 1 \ .$$

Sum-Product Algorithm for finding marginal $\Pr(X_j)$: every variable and factor node computes and sends a message to its remaining neighbor whenever it has received messages from all but one of its neighbors; leaves can send a message immediately. As soon as node X_j has received a message from every one of its neighbors, it does a pointwise multiplication on the messages, and normalizes the result to get $\Pr(X_j)$.

To find all the marginals:

- Arbitrarily pick a root node
- Do one phase of message passing toward root
- Do one phase of message passing away from root

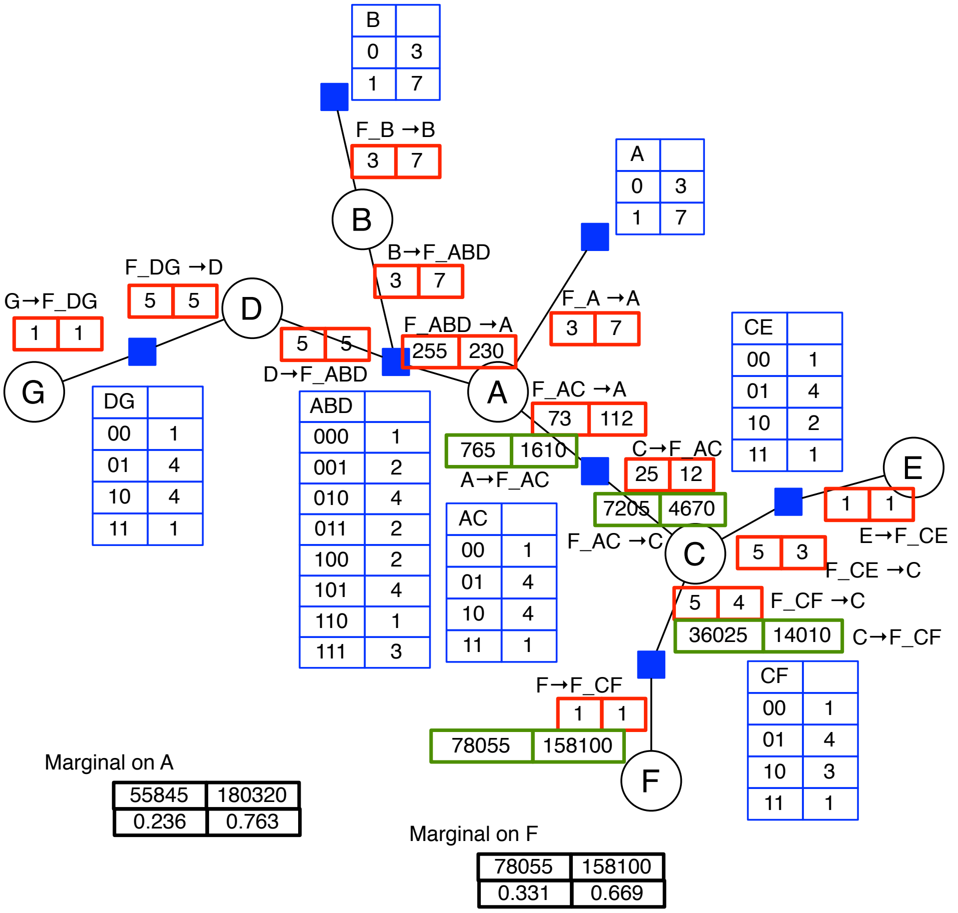
If the original graph was a directed graph, then the marginals we compute will already be normalized; otherwise, the marginals will not be normalized, but since they are over a single variable, it easy to compute $1/Z$ by summing the values of any one of the unnormalized marginals.

Example of sum-product algorithm Figure 2 illustrates the operation of the sum-product algorithm. It contains a factor graph, with the factors shown in blue tables. The messages in red illustrate an “inward pass” toward node A. We’ll work through part if it, starting from the message originating from G.

- The base case for a leaf that is a variable is to pass in a message that is 1 for all values. Recall that a message is mapping from possible values of a variable, in this case, G, to real numbers. In this example, all the variables are binary, so the messages have two numbers, one for variable value 0 and one for variable value 1.
- Now, we compute a message from factor ϕ_{DG} to node D. Recall that

Written F_DG→D in the figure, out of laziness.

$$\mu_{\phi_S \to X_k}(x_k) = \sum_{x_S \setminus x_k} \phi_s(x_s) \prod_{X_m \in X_s \setminus X_k} \mu_{X_m \to \phi_S}(x_m) \ .$$



B	
0	3
1	7

A	
0	3
1	7

DG	
00	1
01	4
10	4
11	1

ABD	
000	1
001	2
010	4
011	2
100	2
101	4
110	1
111	3

AC	
00	1
01	4
10	4
11	1

CE	
00	1
01	4
10	2
11	1

CF	
00	1
01	4
10	3
11	1

Marginal on A

55845	180320
0.236	0.763

Marginal on F

78055	158100
0.331	0.669

Figure 2: Example partial execution of the sum-product algorithm. All variables are binary. Blue boxes are factors; red boxes are message going in to node A; green boxes are messages going out toward F.

In this case,

$$\mu_{\phi_{DG}\to D}(d) = \sum_{g} \phi_{DG}(d,g)\mu_{G\to\phi_{DG}}(g) \enspace .$$

Since the incoming message is all 1, this amounts to summing over all values of G in the factor to get an entry in the outgoing message on D. Because there are only two factors connected to variable D, the message from the factor on B to ϕ_B is the same as the message from B to ϕ_{ABD}.

- Now something interesting happens: we have messages $\mu_{B\to\phi_{ABD}}$ and $\mu_{D\to\phi_{ABD}}$ coming in, so factor ϕ_{ABD} is ready to compute an outgoing message to A. This one is somewhat trickier. We have

  $$\mu_{\phi_{ABD}\to A}(a) = \sum_{bd} \phi_{ABD}(abd)\cdot\mu_{B\to\phi_{ABD}}(b)\cdot\mu_{D\to\phi_{ABD}}(d) \enspace .$$

  One way to think about this computing this is to "multiply the tables". We would do this by taking the table for the factor, and then multiplying the messages into the table: so, for instance, to take the incoming message from B, we would multiply all the entries in the factor table that have $B = 0$ by 3, and all the entries in the factor table that have $B = 1$ by 7. The other factor has a value of 5 for both values, so that results in just multiplying all the table entries by 5. Then, to compute a message to A, we sum all the entries in the product table that have value $A = 0$ and get 255; then sum all entries that have value $A = 1$ and get 230. This is the message from this factor to variable A.

- We will assume that this process has taken place in other parts of the tree, resulting in three messages coming into variable A. Now, we can compute the marginal at A:

  $$\Pr(X_m) \propto \prod_{\phi_l\in\text{neighbors}(X_m)} \mu_{\phi_l\to X_m}(x_m) \enspace .$$

  In this case,

  $$\Pr(A) \propto \mu_{\phi_{ABD}\to A}(a)\cdot\mu_{\phi_A\to A}(a)\cdot\mu_{\phi_{AC}\to A}(a) \enspace .$$

  This gives us an unnormalized potential on A of $(55845, 180320)$, which normalizes to $0.236, 0.763$.

- Now, to illustrate the computation of other marginals, let's see how the computation of $\Pr(F)$ proceeds. The critical thing is that the node A sends a message toward factor ϕ_{AC}, which *is not the marginal on* A. It is computed as usual, taking into account messages from all the factors except ϕ_{AC}. So,

  $$\mu_{A\to\phi_{AC}}(a) = \mu_{\phi_{ABC}\to A}(a)\cdot\mu_{\phi_A\to A}(a) \enspace .$$

- This process continues, generating the messages shown in green, until we get a marginal on F of $(0.497, 0.503)$.

Incorporating evidence Just as in variable elimination, we can add in a extra factors that assign value 1 to observed values of the evidence variables and 0 to non-observed values of the evidence variables, and proceed with the sum-product algorithm as above.

Continuous variables If the distributions are appropriately conjugate, then we can run the same algorithm on graphs of continuous variables. For instance, distributions in which the variables have linear-Gaussian dependence on one another can be handled directly, by replacing sums with integrals.

3.2.3 Finding MAP values

If we want to find the most likely assignment of some variables, we might consider running the sum-product algorithm to get all the marginals and then finding the maximum value in each marginal. Although these values individually maximize the marginals, they do not necessarily constitute a maximizing assignment in the joint probability distribution.

This is a homework exercise.

In the sum product algorithm, we took advantage of the fact that

$$ab + ac = a \cdot (b + c) \ .$$

Now, we will take advantage of a similar relationship for max:

$$\max(ab, ac) = a \cdot \max(b, c) \ .$$

Algebraically, max has the same relationship to multiplication as summation does. That means that our strategy for computing $\max_x \prod_i \phi_i(x)$ can be structurally the same as our strategy for computing $\sum_x \prod_i \phi_i(x)$.

Max-product algorithm If we simply take the sum-product algorithm, change all addition operations to max, and do a single pass inward from leaves to an arbitrarily chosen root note, then we will compute $\max_x \Pr(x)$.

Max-sum algorithm Multiplying all those small probabilities can lead to serious loss of precision; to make the computation better conditioned, we can work with log probability values. Because log is monotonic, it preserves the maximum, so:

$$\log \max_x \prod_i \phi_i(x) = \max_x \log \prod_i \phi_i(x) = \max_x \sum_i \log \phi_i(x) \ .$$

The distributive property is preserved because

$$\max(a + b, a + c) = a + \max(a, c) \ .$$

So, if we change the max-product algorithm by replacing products of factor values by sums of logs of factor values, we get the *max-sum* algorithm, which has the following messages:

$$\mu_{\phi_S \to X_k}(x_k) = \max_{x_S \setminus x_k} \log \phi_s(x_s) + \sum_{X_m \in X_s \setminus X_k} \mu_{X_m \to \phi_S}(x_m) \ .$$

$$\mu_{X_m \to \phi_S}(x_m) = \sum_{\phi_l \in \text{neighbors}(X_m) \setminus \phi_S} \mu_{\phi_l \to X_m}(x_m) \ .$$

Base case if ϕ is a leaf:

$$\mu_{\phi \to X_i}(x_i) = \log \phi(x_i) \ .$$

Base case if X_i is a leaf:

$$\mu_{X_i \to \phi}(x_i) = 0 \ .$$

Backtracking Now, what we might really want to find is the entire maximizing assignment, not just its probability. Unfortunately, we can't do it using the straightforward extension of sum-product and doing an "outward" pass of message propagation through the algorithm, in case there are multiple joint assignments with the same maximizing probability that have different assignments.

Not to be confused with backtracking search...

This is sometimes called "decoding" and has a connection to coding theory.

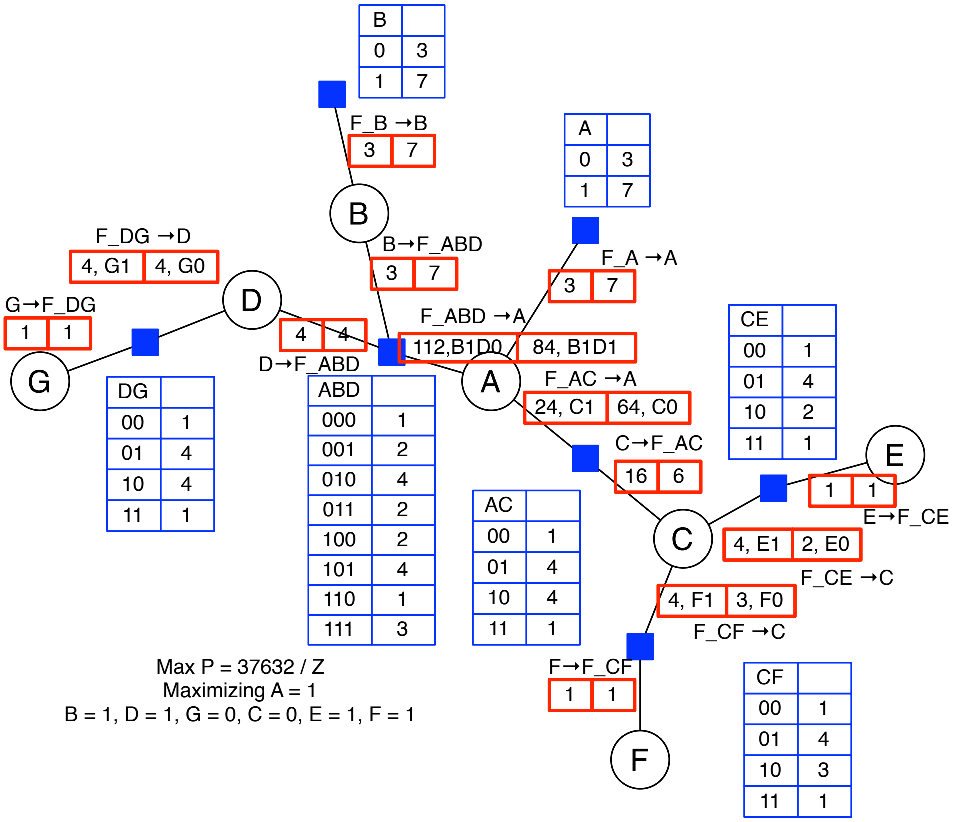


Figure 3: Example execution of the max-product algorithm. All variables are binary. Blue boxes are factors; red boxes are message going in to node A, and also include the local assignments that were selected to compute a particular maximum value; maximizing assignment shown in black text at the bottom.

We begin with an inward pass, but with a bit of extra bookkeeping. When we compute inward message from each clique,

$$\mu_{\phi_S \to X_k}(x_k) = \max_{x_S \setminus x_k} \log \phi_s(x_s) + \sum_{X_m \in X_s \setminus X_k} \mu_{X_m \to \phi_S}(x_m) \ .$$

we keep track, for each value of x_k, of a set $Y_S^*(x_k)$ of possible maximizing joint assignments to $X_S \setminus X_k$, which were responsible for the max value in the message.

Then, for whatever node X_j we decided to use as the root in the first pass of max-sum, we know that $x_j^* = \arg\max_{x_j} \Pr(X_j = x_j)$ is a component of some MAP assignment. Now, we begin an outward pass. Instead of passing a probability message into a clique ϕ_S, we pass in the maximizing value x_k^* of the variable that was missing when it was evaluated. Now, that cliques selects any member of $Y_S^*(x_k)$, which provides maximizing assignments for the variables $X_S \setminus X_k$. These maximizing assignments are propagated to any cliques that are neighbors of variables in $X_S \setminus X_k$, and so on, out to the trees.

Example of max-product algorithm The figure 3 shows an example execution of the max-product algorithm. We will walk through some parts of the computation.

We didn't take logs and do max-sum here, in order to keep the numbers intuitively understandable; but in a real example, it's much more numerically stable to do max-sum.

- This time, let's start from node F. The message $\mu_{F \to \phi_{CF}}$ is the base case, $(1, 1)$. We multiply it into the factor ϕ_{CF} . Now, instead of summing out values of F, we maxi-

Which has no effect because it is $(1, 1)$.

mize over them. The result is a message on C, which says that for $C = 0$ we an attain a value of 4, if $F = 1$, and that for $C = 1$ we can attain a value of 3, if $F = 0$.

- At node C, we multiply the incoming messages $(4, 3)$ and $(4, 2)$, to get an outgoing message $\mu_{C \to \phi_{AC}}$ of (16, 6).
- We keep going...let's see what finally happens at node A. We have incoming messages on all the arcs, and just do a pointwise multiply to get a factor on A of $(8064, 37632)$. This potential **is not** a marginal distribution on A. But we can conclude that value of A in an assignment of values to all the variables that maximizes $\Pr(X)$ is $A = 1$.

Because that value is higher in the potential.

- Now, we can do the "backtracking" to find the rest of the assignment. We see that, for $A = 1$, to get the value 84 from ϕ_{ABD}, we need $B = 1$ and $D = 1$.
- If $D = 1$, then to get value 4 from ϕ_{DG}, we need to have $G = 0$.
- Similarly, in the other part of the network, we see we need $C = 0$, which forces $E = 1$ and $F = 1$.

3.3 Converting graphs to trees

The message-passing algorithms are only well-defined on factor graphs that are trees. If we want to do exact inference on a graph that is not a tree, we must convert it into a tree. It is possible to look at the sequence of factors that get generated during variable elimination and use them to create a tree of cliques, which is sometimes called a *junction tree*. Then, there is a message passing algorithm on the junction tree that can be used to compute all the marginals. It is exponential in the tree width (as is variable elimination).

4 Approximate Inference

If we have a graph with a large tree-width, or with a mixture of distributions that makes it impossible to represent intermediate messages, then we have to fall back on approximate inference methods. We will explore three different strategies.

4.1 Loopy BP

One easy approximation method, in models for which belief propagation is appropriate (e.g., all discrete or linear Gaussian models), but where the factor graph is not a tree, is to apply the message passing algorithm. Now, instead of just doing two passes, we could continue to apply it, iteratively.

- This algorithm will not always converge. It is possible to increase the chance of convergence by, instead of computing a new message each time through, to average the old message with the newly-calculated message, and propagate the average instead.
- If it does converge, it might converge to the wrong answer. In particular, it can become overconfident about certain values because information that some evidence yields about a variable might be "double counted" if there are multiple paths through the graph from the evidence to the node in question.

4.2 Variational methods

Another strategy, with much wider applicability is to use *variational methods*. Assume that we are trying to compute some distribution $p * (x)$, but it is intractable. We might, instead, try to find an approximation from some other family of distributions Q. We would like to pick Q so that

- It is in some way simpler than P, so that it will be computationally easier to work with;
- It can come close to representing p^* in the aspects that are important to us.

This approach is also useful in Bayesian models where it is difficult to directly construct an exact representation of a posterior, for example, when it is non-conjugate. Such applications are often called *variational Bayes*.

So, we will treat inference as an *optimization* problem: we will seek to find $q \in Q$ that is as close as possible to p^*. A typical measure on the distance between two distributions is the Kullback-Liebler divergence:

It is called a divergence because it is not a proper distance metric: it is not symmetric and doesn't satisfy the triangle inequality.

$$\mathrm{KL}(p \parallel q) = \sum_x p(x) \log \frac{p(x)}{q(x)} \quad .$$

It ranges from 0 (if the distributions are equal) to infinity (if $q(x)$ assigns 0 probability to an element that has positive probability in p).

It might make sense to seek $q^* = \arg\min_q \mathrm{KL}(p^* \parallel q)$, but because we assume that p^* is hard to evaluate, this will be also difficult. Instead, we will try to find

$$q^* = \arg\min_q \mathrm{KL}(q \parallel p^*) = \sum_x q(x) \log \frac{q(x)}{p^*(x)} \quad .$$

Even evaluating $p^*(x)$ at a point might be hard because of the normalizing constant, Z. So, letting $\tilde{p}(x) = Zp^*(x)$, we will try to minimize

$$\begin{aligned} J(q) &= \mathrm{KL}(q \parallel \tilde{p}) \\ &= \sum_x q(x) \log \frac{q(x)}{\tilde{p}(x)} \\ &= \sum_x q(x) \log \frac{q(x)}{Zp^*(x)} \\ &= \sum_x q(x) \log \frac{q(x)}{p^*(x)} - \log Z \\ &= \mathrm{KL}(q \parallel p^*) - \log Z \end{aligned}$$

Since Z is a constant, if we minimize J, we will be forcing q to be close to p^*.

Mean Field approximation The *mean field* approximation is a version of variational inference in which we pick Q to be the family of completely factored distributions: that is, where

$$q(x) = \prod_i q_i(x_i) \quad .$$

So, to optimize J, we will optimize the parameters of each individual marginal to come as close as possible to the joint distribution.

We will minimize this by *coordinate descent*, in which we adjust one parameter to minimize the objective, holding all the others constant. Let $E_{-j}[f(x)]$ be the expectation over all the variables except for x_j.

We will *maximize*

$$L(q) = -J(q) = \sum_x q(x) \log \frac{\tilde{p}(x)}{q(x)} \quad .$$

If we treat all terms that do not involve q_j as constants, we get

Derivation in Murphy section 31.3.1.

$$L(q_j) = \sum_{x_j} q_j(x_j) \log f_j(x_j) - \sum_{x_j} q_j(x_j) \log q_j(x_j) + c \quad ,$$

where

$$\log f_j(x_j) = \sum_{x_{-j}} \prod_{i \neq j} q_i(x_i) \log \tilde{p}(x) = E_{-j} \left(\log \tilde{p}(x)\right) \quad .$$

You can see this as

$$L(q_j) = -KL(q_j \parallel f_j), \quad ,$$

which is maximized by setting $q_j = f_j$:

$$q_j(x_j) = \frac{1}{Z_j} \exp\left(E_{-j}(\log \tilde{p}(x))\right) \quad .$$

We can usually ignore the normalization constant, so we can update

$$\log q_j(x_j) = E_{-j}(\log \tilde{p}(x)) \quad .$$

This a set of equations that may be hard to solve, but they can form the basis of an iterative algorithm; it is guaranteed to converge, because L is convex with respect to each of the factors q_i.

This process is particularly nice in graphical models because computing $E_{-j}(\log \tilde{p}(x))$ really only depends on the Markov blanket of the node j.You take an expectation over all assignments of values to the variables in the Markov blanket, according to the distribution of values assigned to them by the current q_i parameters.

The rest of the factors in the distribution are constant from the perspective of x_j

$$\log q_j(x_j) = \sum_{x_{mb(j)}} \left(\prod_{i \in mb(j)} q_i(x_i) \right) \sum_{\phi_s \in \text{neighbors}(X_j)} \log \phi_s(x_{mb(j)} \cup \{x_j\}) \quad ;$$

where $x_{mb(j)}$ is an assignment of values to variables in the Markov blanket of X_j.

Intuitively, we:

- Start with some assignment to the marginal of each variable, q_i;
- Select a variable X_j to update;
- Compute its distribution, using the distributions q_i on variables i in the Markov blanket, together with the factors ϕ_S that involve variable X_j;
- Assign q_j to be that marginal on X_j.

4.3 Sampling

We can estimate a distribution by drawing samples from it, and then doing density estimation: fit a Gaussian, or count, or fit a histogram, etc.

We can estimate an expectation by observing that

$$E[f(x)] = \int_x f(x) \Pr(x) \approx \frac{1}{n} \sum_{i=1}^{n} f(x^{(i)}) \quad ,$$

for $x^{(i)}$ drawn from Pr, for large enough n.

In a directed graphical model, it is easy to sample from $\Pr(X)$:

This method is sometimes called *ancestral sampling*.

- Sample from the "root" nodes that have no parents;
- For each additional node, draw a sample from the conditional distribution, conditioned on the sampled values for the parents.

Given a set of samples of the joint distribution, you can obtain samples of any marginal just by looking at the relevant dimensions of the joint samples.

In an undirected model, even sampling from the joint is difficult: we will concentrate on directed models for a while, and return to undirected models in the context of Monte-Carlo methods and Gibbs sampling.

An alternative, but potentially hairy, strategy is to convert the undirected model to a directed model and proceed as described here.

4.3.1 Rejection sampling

What if we are in a directed model and we wish to estimate $\Pr(Y \mid E = e)$ for some set of variables $Y \subset X$? A valid strategy is to draw samples from the joint distribution, throw away any of them that do not have $E = e$, and then observe the empirical distribution over Y in that set of samples. This is great if $\Pr(E = e)$ is reasonably high, but if $E = e$ is a rare event, this is very inefficient.

4.3.2 Importance sampling

It feels like maybe we could just force the evidence variables to have the desired values, and combine that with ancestral sampling. But then we wouldn't be drawing from the correct distribution. We'll do this in a more subtle way, called *likelihood weighting*, which generates a set of samples that are weighted: to estimate our final quantity or distribution, we will have to work with the weighted samples.

To generate a weighted sample $\hat{x}^{(j)}$:

- Set $w_j = 1$
- Do ancestral sampling, starting from the roots, to construct $\hat{x}$
- Whenever you account a variable $X_i \in E$:
  - Set $\hat{x}_i = e_i$
  - Let $w_j = w_j \cdot \Pr(e_i \mid \text{parents of } X_i \text{ in } \hat{x})$

Now, to compute the resulting distribution, we use weights

$$\hat{\Pr}(X_i = x_i \mid E = e) = \frac{\sum_{j=1}^{m} w_j I(\hat{x}_i^{(j)} = x_i)}{\sum_{j=1}^{m} w_j} \quad .$$

This is an instance of a method called *importance sampling*, in which we wish we were able to draw samples from P, but instead we are able to draw them from Q. We can take advantage of the following relationship:

$$E_{P(X)}[f(X)] = E_{Q(X)}\left[f(X)\frac{P(X)}{Q(X)}\right] \sum_{x} Q(x) f(x) \frac{P(X)}{Q(X)} = \sum_{x} f(x) P(x) \quad .$$

If we draw samples $x^{(j)}$ from Q, then

$$E_{P(X)}[f(X)] \approx \frac{1}{m} \sum_{j=1}^{m} f(x^{(j)}) \frac{P(x^{(j)})}{Q(x^{(j)})} \quad .$$

In likelihood weighting, the Q distribution is the one we get when we force all of the evidence variables to have the specified values. This is sometimes called the *mutilated network* distribution.

$$w(x) = \frac{P(x)}{Q(x)} = \frac{\prod_i \Pr(x_i \mid pa(x_i), e)}{\prod_{i \notin E} \Pr(x_i \mid pa(x_i), e)} = \prod_{i \in E} \Pr(e_i \mid pa(e_i)) \ \ .$$

4.3.3 Gibbs sampling

One problem with likelihood weighting is that evidence only affects the sampling for descendant nodes: if the evidence is at a leaf, then it doesn't really help. *Gibbs sampling* is an instance of a general class of sequential sampling methods called *Markov-chain Monte Carlo* methods. The idea is that we will generate a sequence of samples:

- The samples we generate are no longer independent;
- After we run the process for a certain amount of time (called the "burn-in period") the samples will be IID;
- The distribution they are drawn from is the desired conditional distribution;
- May not work if there are 0 values in the factors. distributions.

It is easiest to think of this algorithm on an undirected graph:

- Start by generating an initial sample $x^{(0)}$ using forward sampling
- For $i = 1, \ldots, n$ do
  - $x^{(i)} = x^{(i-1)}$
  - For $j = 1 \ldots, d$ do:
    - Sample $x_j^{(i)}$ from $P_\phi(x_i \mid \text{MarkovBlanket}(x_j^{(i)}))$

There are variations in which we only change one value at a time.

Note that this sampling takes the child values into account. Remember that in a directed model, the Markov blanket consists of parents (Z), children (C), and other parents of the children (O).

$$\begin{aligned} \Pr(X \mid Z, C, O) &\propto \Pr(C \mid Z, X, O) P(X \mid Z, O) \\ &= \Pr(C \mid X, O) P(X \mid Z) \end{aligned}$$

Effective, but practically tricky:

- How long to "burn in"?
- Use every sample, or skip some?

5 Parameter estimation

Finally! We get to do learning! There are two parts to learning a graphical model: learning the graph structure and learning the parameters. We will start by thinking about how to learn the graph structure, assuming the parameters are fixed.

5.1 Completely observed data

The easiest case is when we observe values for all the variables in our model:

$$\mathcal{D} = \{x^{(1)}, \ldots x^{(n)}) \ .$$

In this case, our job is simple parameter estimation of the kind that we did in the beginning of the term. One can find maximum-likelihood estimates or take a Bayesian approach. It is common to assume a $\text{Beta}(1,1)$ or uniform Dirichlet prior, and then effectively use the Laplace correction in estimating the parameter values.

5.2 Latent variable models and EM

Things start to get interesting and tricky when all the data are not available: it might be that you're just missing some observations of some attributes in the data set or that entire nodes are *latent*: that is, never observed. We will explore an algorithm called EM for *expectation-maximization* for maximum likelihood parameter estimation when there are unobserved values in the model.

We'll start with a very simple problem, in which single attribute of a single data set is missing. There are two attributes, A and B, and this is our data set, $\mathcal{D}$:

i	A	B	
1	1	1	
2	1	1	
3	0	0	
4	0	0	
5	0	0	
6	0	H	***missing **
7	0	1	
8	1	0	

Assume the data is *missing completely at random* (MCAR): that is, that the fact that it is missing is independent of its value.

Our goal is to estimate $\Pr(A, B)$ from this data. We'd really like to find the maximum-likelihood parameter values, if we can. The likelihood is

$$\mathcal{L}(\theta) = \log \Pr(\mathcal{D}; \theta) = \log\left(\Pr(\mathcal{D}, H = 0; \theta) + \Pr(\mathcal{D}, H = 1; \theta)\right) \ .$$

Naive strategy 1 Maybe we're lazy. We could just ignore $x^{(6)}$ all together, and estimate the parameters:

$$\widehat{\theta}^1 = \begin{pmatrix} 3/7 & 1/7 \\ 1/7 & 2/7 \end{pmatrix} = \begin{pmatrix} .429 & .143 \\ .143 & .285 \end{pmatrix}$$

If we do that, then

$$\begin{aligned} \mathcal{L}(\widehat{\theta}^1) &= \log\left(\Pr(00\ ;\ \widehat{\theta}^1)\prod_{i\neq 6}\Pr(x^i\ ;\ \widehat{\theta}^1) + \Pr(01\ ;\ \widehat{\theta}^1)\prod_{i\neq 6}\Pr(x^i\ ;\ \widehat{\theta}^1)\right) \\ &= 3\log 0.429 + 2\log 0.143 + 2\log 0.285 + \log(0.429 + 0.143) \\ &= -9.498 \end{aligned}$$

Naive strategy 1 Let H be the 'best' value it could have, that is to make the log likelihood as large as possible. That value is 0. So, then we'd have

$$\widehat{\theta}^2 = \begin{pmatrix} .5 & .125 \\ .125 & .25 \end{pmatrix}$$

and

$$\mathcal{L}(\widehat{\theta}^2) = -9.481 \ .$$

That's a little better!

Non-naive strategy: optimize likelihood directly

$$\widehat{\theta} = \arg\max_{\theta} \mathcal{L}(\theta) \ .$$

We can do this with gradient methods, but it gets tricky because of constraint that $\widehat{\theta}$ be a valid probability distribution. Instead we'll explore a new algorithm.

5.2.1 EM Algorithm

If we have a parameter estimation problem in which there are some variables, H, which are unobserved, we approach the problem of finding a distribution, $\tilde{P}$, on H together with parameters θ. We will do this by *coordinate ascent*, alternating between holding the first constant while optimizing the second and holding the second constant while optimizing the first. Define

$$\mathcal{L}(\tilde{P}, \theta) = \sum_h \tilde{P}(h) \log \frac{\Pr(\mathcal{D}, h \ ; \ \theta)}{\tilde{P}(h)}$$

and

$$P_H(h) = \Pr(h \mid \mathcal{D} \ ; \ \theta) \ .$$

Then, for any $\tilde{P}$

$$\mathcal{L}(\theta) = \log \Pr(\mathcal{D} \ ; \ \theta) = \mathcal{L}(\tilde{P}, \theta) + \mathrm{KL}(\tilde{P} \parallel P_H) \ .$$

KL divergence is defined

$$\mathrm{KL}(\tilde{P} \parallel P_h) = -\sum_h \tilde{P}(h) \log \frac{P(h \mid \mathcal{D} \ ; \ \theta)}{\tilde{P}(h)} \ .$$

KL is always $\geqslant 0$, with value 0 when distributions are equal.

Bound $\mathcal{L}(\tilde{P}, \theta)$ is a *lower bound* on $\mathcal{L}(\theta)$ for any $\tilde{P}$. So, we will try to maximize $\mathcal{L}(\tilde{P}, \theta)$ by coordinate ascent.

Algorithm

- Initialize θ^{old} arbitrarily

But the result of the algorithm will be sensitive to this choice.

- **E step**: Hold θ^{old} constant; select $\tilde{P}$ to maximize $\mathcal{L}(\tilde{P}, \theta^{\text{old}})$.
  This happens when $\mathrm{KL}(\tilde{P}, P_H) = 0$, so $\tilde{P} = P_H$, so,

  $$\tilde{P}(h) := \Pr(H = h \mid \mathcal{D} \ ; \ \theta^{\text{old}}) \ .$$

- **M step**: Hold $\tilde{P}$ constant; select θ^{new} to maximize $\mathcal{L}(\tilde{P}, \theta^{\text{new}})$.

  $$\theta^{\text{new}} := \arg\max_{\theta} \sum_h \tilde{P}(h) \log \Pr(\mathcal{D}, H = h \ ; \ \theta) \ .$$

  Now, we know that unless $\theta^{\text{new}} = \theta^{\text{old}}$, $\mathcal{L}(\tilde{P}, \theta)$ has increased and so has $\mathrm{KL}(\tilde{P}, P_H)$; so $\mathcal{L}(\theta)$ has increased as well.

- If $\theta^{\text{old}} \approx \theta^{\text{new}}$ or $\mathcal{L}(\theta^{\text{old}}) \approx \mathcal{L}(\theta^{\text{new}})$, then terminate.

- Else, repeat M step and E step.

Properties of EM

- Monotonic convergence to a local optimum of $\mathcal{L}(\theta)$
- Result depends on initialization
- E and M steps usually easier than direct optimization of log likelihood
- Extends easily to Bayesian MAP case: find (local)

  $$\arg\max_{\theta} \Pr(\theta \mid \mathcal{D}) \ ,$$

  taking prior $\Pr(\theta)$ into account.
- Sometimes the M step can be computationally difficult. Generalized EM can still converge if M step generates an improvement in $\mathcal{L}$ but not necessarily the maximum.

Back to the missing data example

- Guess

  $$\theta_0 = \begin{pmatrix} .25 & .25 \\ .25 & .25 \end{pmatrix}$$

- **E Step**

  $$\tilde{P}(H = 1) = \Pr(H = 1 \mid \mathcal{D}\ ;\ \theta_0) = \Pr(H = 1 \mid x^{(6)}\ ;\ \theta_0) = \Pr(B = 1 \mid A = 0\ ;\ \theta_0) = 0.5$$

- **M Step**

  $$\begin{aligned} \theta_1 &= \arg\max_{\theta} \left(0.5 \log \Pr(\mathcal{D}, H = 0\ ;\ \theta) + 0.5 \log \Pr(\mathcal{D}, H = 1\ ;\ \theta)\right) \\ &= \begin{pmatrix} 7/16 & 3/16 \\ 2/16 & 4/16 \end{pmatrix} \end{aligned}$$

  This step is not immediately obvious: to derive it, we need to take the derivative with respect to each of the parameters, set to 0, and solve for θ. We find that we can treat the estimation problem as one in which we have a data item for each possible value of H, weighted by the probability that H has that value. We get such a decomposition because, for each particular value of H, only one of the parameter estimates is affected.

We get the same result by doing estimation as usual, but treating $\Pr(H)$ as giving us fractional counts on both data cases:

i	A	B	count = P~(H)
1	1	1	
2	1	1	
3	0	0	
4	0	0	
5	0	0	
6a	0	0	0.5
6b	0	1	0.5
7	0	1	
8	1	0	

On subsequent EM iterations, we have $\mathcal{L}(\theta) = -10.39, -9.47, -9.4524, -9.4514, \ldots$.

5.2.2 Bayesian networks with hidden nodes

Why would we consider learning a Bayes net with a completely hidden node? Because sometimes causal theories that postulate a hidden, shared cause are simpler. Compare:

- Completely connected graph on 4 highly dependent observable variables
- Graph with hidden H as parent and 4 observable variables that are conditionally independent given H.

How can we estimate CPTs when we don't know the values of H? With two observable variables, one row of our data looks like:

$$x_1^{(i)}, x_2^{(i)}, h^{(i)} \ .$$

- **Guess** θ_0.
  Now, assuming the rows in our data table are independent, we we can find

  $$\Pr(h) = \Pr(h \mid x \ ; \ \theta) = \prod_i \Pr(h^{(i)} \mid x^{(i)} \ ; \ \theta) \ .$$

  Computing this is a Bayes net inference problem we know how to solve, no matter where h is in the network (or if there are multiple hidden variables).

- **M Step** Turn

  $$x_1^{(i)}, x_2^{(i)}, h^{(i)} \ .$$

  into

  $$\begin{array}{lll} x_1^{(i)}, x_2^{(i)}, 0 & \text{with weight} & \Pr(h^{(i)} = 0 \mid x^{(i)} \ ; \ \theta) \\ x_1^{(i)}, x_2^{(i)}, 1 & \text{with weight} & \Pr(h^{(i)} = 1 \mid x^{(i)} \ ; \ \theta) \end{array}$$

  These weights are sometimes called *responsibilities*.

- **E step:** Estimate θ using weighted data counts.

  As in the simple case, the fact that we can do estimation with the expected counts under $\Pr(h)$ is derivable by finding the α that maximizes the expectation, under $\Pr(h)$ of the log of the complete data likelihood.

5.2.3 Gaussian mixtures

One particular latent-variable Bayesian network that is of particular interest is one in which there is a latent discrete variable Z with values in $\{1, \ldots, k\}$ and an observable continuous variable X with values in $\mathbb{R}^d$. The network structure is $Z \to X$. This model can be used for density estimation or clustering, depending on what we're interested in doing with our data.

To make some things come out more beautifully, we will let $Z = (Z_1, \ldots, Z_k)$ with $Z_j \in \{0, 1\}$, with the constraint that $\sum_j Z_j = 1$; that is, we will have a single indicator variable for each cluster, so that $z_j^{(i)} =$ means that example i was drawn from Gaussian component j.

$$\begin{array}{rcl} Z & \sim & \text{Multinomial}(\pi_1, \ldots, \pi_k) \\ X \mid Z_j = 1 & \sim & \text{Normal}(\mu_j, \Sigma_j) \end{array}$$

If we knew the values of Z*, life would be easy.* The *complete-data likelihood* is:

$$\begin{aligned}
\Pr(X, Z \,;\, \mu, \Sigma, \pi) &= \prod_{i=1}^{n} \Pr(x^{(i)}, z^{(i)} \,;\, \mu, \Sigma, \pi) \\
&= \prod_{i=1}^{n} \Pr(z^{(i)} \,;\, \pi) \Pr(x^{(i)} \,;\, z^{(i)}\mu, \Sigma) \\
&= \prod_{i=1}^{n} \prod_{j=1}^{k} \pi_j^{z_j^{(i)}} \mathcal{N}(x^{(i)} \,;\, \mu_j, \Sigma_j)^{z_j^{(i)}}
\end{aligned}$$

Log likelihood

$$\mathcal{L}(\mu, \Sigma, \pi) = \sum_{i=1}^{n} \sum_{j=1}^{k} z_k^{(i)} (\log \pi_j + \log \mathcal{N}(x^{(i)} \,;\, \mu_j, \Sigma_j)) \quad .$$

It's clear that if we knew the $z_j^{(i)}$ it would be trivial to find the maximizing parameters using regular parameter estimation. In EM, we have to take the expectation over the Z.

Algorithm

- Choose initial π^{old}, μ^{old}, Σ^{old}
- **E Step** Compute *responsibilities*, which are probabilities of the hidden variables having value 1:
$$\gamma(z_k^{(i)}) = E[z_j^{(i)}] = \frac{\pi_j \mathcal{N}(x^{(i)} \,;\, \mu_j, \Sigma_j)}{\sum_{l=1}^{k} \pi_l \mathcal{N}(x^{(i)} \,;\, \mu_l, \Sigma_l)} \quad .$$
- **M Step** Re-estimate parameters using the fractional responsibilities as weights on the data.
$$\begin{aligned}
\mu_j^{\text{new}} &= \frac{1}{n_j} \sum_{i=1}^{n} \gamma(z_j^{(i)}) x^{(i)} \\
\Sigma_j^{\text{new}} &= \frac{1}{n_j} \sum_{i=1}^{n} \gamma(z_j^{(i)}) (x^{(i)} - \mu_j^{\text{new}})(x^{(i)} - \mu_j^{\text{new}})^T \\
\pi_j^{\text{new}} &= \frac{n_j}{n} \\
n_j &= \sum_{i=1}^{n} \gamma(z_j^{(i)})
\end{aligned}$$

Some things to worry about:

- EM only goes to a local optimum, which depends on initialization
- The model is not *identifiable*: that is, there may be many "best" models in the likelihood sense. A particular issue is the assignment of the class variables: the likelihood is the same if they are interchanged.
- The optimization problem is ill-formed: the model can get infinite likelihood score if it assigns a single point to a cluster, which then gives it variance approaching zero. In practice, it is usually necessary to implement some special safeguards to keep this from happening.

5.2.4 Exponential family and expected sufficient statistics

A *statistic* is a function $T(\mathcal{D})$ of the data. A statistic is *sufficient* for parameter θ if $\mathcal{D}$ is conditionally independent of θ given $T(\mathcal{D})$. That is, $T(\mathcal{D})$ captures everything that is relevant about the data for θ.

This exposition is following from `people.missouristate.edu/songfengzheng/Teaching/MTH541.`

You don't have to follow this derivation, but the punch line is important.

Factorization theorem $T(\mathcal{D})$ is a sufficient statistic for θ iff $\Pr(\mathcal{D} \mid \theta)$ can always be factorized as

$$\Pr(\mathcal{D} \mid \theta) = u(\mathcal{D})v(T(\mathcal{D}), \theta) \ \ ,$$

where u and v are nonnegative functions.

Now, we can show that the maximum likelihood estimator depends only on the sufficient statistic:

$$\begin{aligned}\theta_{ML} &= \arg\max_\theta \Pr(\mathcal{D} \mid \theta)\\ &= \arg\max_\theta u(\mathcal{D})v(T(\mathcal{D}), \theta)\\ &= \arg\max_\theta v(T(\mathcal{D}), \theta)\end{aligned}$$

Distributions in the exponential family have sufficient statistics of the same dimension as the parameter space, regardless of sample size. One-parameter members of the exponential family have

$$\Pr(x \mid \theta) = \exp(c(\theta)T(x) + d(\theta) + S(x)) \ \ .$$

So

$$\begin{aligned}\Pr(\mathcal{D} \mid \theta) &= \prod_{i=1}^{n} \exp(c(\theta)T(x^{(i)}) + d(\theta) + S(x^{(i)}))\\ &= \exp\left(\sum_{i=1}^{n} c(\theta)T(x^{(i)}) + nd(\theta)\right) \exp\left(\sum_{i=1}^{n} S(x^{(i)})\right)\end{aligned}$$

So, $\sum_{i=1}^{n} T(x^{(i)})$ is a sufficient statistic (by the factorization theorem). And (taking the log) we can write the maximum-likelihood estimator as

$$\theta_{ML} = \arg\max_\theta c(\theta)T(\mathcal{D}) + nd(\theta) \ \ .$$

In the M step of EM, when Pr is in the exponential family, then we want

$$\begin{aligned}\arg\max_\theta \mathcal{L}(\hat{P}, \theta) &= \arg\max_\theta \sum_Z \hat{P}(Z) \log \Pr(X, Z \mid \theta)\\ &= \arg\max_\theta \sum_Z \hat{P}(Z) \log \exp\left(c(\theta)T(X, Z) + nd(\theta)\right)\\ &= \arg\max_\theta \sum_Z \hat{P}(Z)(c(\theta)T(X, Z) + nd(\theta))\\ &= \arg\max_\theta c(\theta) \sum_Z \hat{P}(Z)T(X, Z) + nd(\theta)\\ &= \arg\max_\theta c(\theta)\bar{T}(X, Z) + nd(\theta)\end{aligned}$$

where $\bar{T}(X, Z) = \sum_Z \hat{P}(Z)T(X, Z)$. *This estimator has the same form as the maximum likelihood estimator, but using the expected value of the sufficient statistics in place of the actual sufficient statistics (which are unknown because we don't know the values of* Z*).*

5.3 Parameter estimation in undirected models

This is generally a very difficult problem: all methods are some version of gradient descent. The reason it is hard is that the computation of the gradient at each step requires a full inference step in the graphical model. We won't go into it further.

6 Structure learning

Now we know how to estimate the parameters in a graphical model given the structure. How can we estimate the structure? There are two general strategies:

- Do a pre-processing pass in which some measure of the correlations between pairs of variables is computed; then build a structure based on the strength of those correlations.
- Search directly in the space of network structures: for each structure, finding the maximum-likelihood parameters and score the structure using a weighted combination of the likelihood of the training data and a penalty for model complexity. There is a beautiful Bayesian scoring method we will also consider.

6.1 Finding the best tree-structured model

Define a *tree-structured network* to be a directed graphical model in which no node has more than one parent. There is a polynomial-time algorithm for finding the best tree-structured model, given a data set. It is due to Chow and Liu.

This is a subclass of polytrees, in which a node may have more than one parent.

The joint distribution for any tree-structured model can be written:

$$\Pr(X) = \prod_j \Pr(X_j) \prod_{j,k} \frac{\Pr(X_j, X_k)}{\Pr(X_j)\Pr(X_k)} \quad .$$

Verify this for yourself by example: draw any tree-structured directed graphical model, write out the joint distribution in terms of conditional probabilities, use the definition of conditional probability to turn each into a joint divided by a marginal, and cancel.

The log likelihood for a tree is:

$$\Pr(\mathcal{D}; \theta, T) = \sum_t \sum_k N_{tk} \log \Pr(X_t = k; \theta) + \sum_{s,t} \sum_{j,k} N_{stjk} \log \frac{\Pr(X_s = j, X_t = k; \theta)}{\Pr(X_s = j; \theta)\Pr(X_t = k; \theta)} \quad ,$$

where N_{stjk} is the number of times $X_s = j$ and $X_t = k$. We'll define the *empirical distribution* to be the obvious estimates of these quantities from the data:

$$\begin{aligned} \hat{P}(X_t = k) &= \frac{N_{tk}}{N} \\ \hat{P}(X_s = j, X_t = k) &= \frac{N_{stjk}}{N} \end{aligned}$$

So,

$$\frac{\Pr(\mathcal{D}; \theta, T)}{N} = \sum_t \sum_k \hat{P}(X_t = k) \log \hat{P}(X_t = k) + \sum_{s,t} \hat{I}(X_s, X_t) \quad ,$$

where $\hat{I}(X_s, X_t)$ is the mutual information between X_s and X_t in the empirical distribution; that is

$$\hat{I}(X_s, X_t) = \sum_j \sum_k \hat{P}(X_s = j, X_t = k) \log \frac{\hat{P}(X_s = j, X_t = k)}{\hat{P}(X_s = j)\hat{P}(X_t = k)} \quad .$$

The first term in the log likelihood is independent of the structure; so when choosing structure, we only need to worry about the second term. So, the maximum likelihood tree structure is the maximum weighted spanning tree, where the edge weights are the pairwise empirical mutual informations. Finding the MST can run in $O(E \log V)$ time where $E = D^2$ is the number of edges and D is the number of nodes. So, the overall running time is $O(ND^2 + D^2 \log D)$.

6.2 Structure search

If we want a general, non-tree structure, then there's no straightforward method. We typically just do direct local search in the space of structures.

- Pick an initial graph, G, possibly by running ChowLiu
- Loop
  - Propose several local changes to the graph: add an arc, delete an arc, reverse an arc
  - For each proposed graph, G', compute the maximum likelihood score:

    $$\text{scoreML}(G'; D) = \max_\theta \Pr(\mathcal{D} \mid \theta, G') \ .$$

  - Depending on the search strategy, select a G' and set $G = G'$

Reversing an arc can be accomplished by deleting, and then adding a new one back in the other direction; but deleting the arc might decrease the score, preventing this move from being considered.

Some points about this algorithm:

- Note that there are many graph structures that represent equivalent sets of conditional independence assumptions;cut down the search space by only considering on element of each equivalence class of graph structures
- The simplest search strategy is to always select the G' with the highest score, and to terminate if the new score is less than or equal to the old one.
- Could be better to use some form of stochastic search, such as simulated annealing, that sometimes takes a "downhill" step, in order to avoid local optima.
- The scoring function is *decomposable*:

$$\begin{aligned}
\text{scoreML}(G; D) &= \max_\theta \Pr(\mathcal{D}; \theta, G) \\
&= \max_\theta \prod_{i=1}^{n} \Pr(x^{(i)}; \theta, G) \\
&= \max_\theta \prod_{i=1}^{n} \prod_{j=1}^{D} \Pr(x_j^{(i)} \mid \text{Pa}(x_j, G); \theta, G) \\
&= \max_\theta \prod_{j=1}^{D} \prod_{i=1}^{n} \Pr(x_j^{(i)} \mid \text{Pa}(x_j, G); \theta_j, G) \\
&= \max_\theta \sum_{j=1}^{D} \sum_{i=1}^{n} \log \Pr(x_j^{(i)} \mid \text{Pa}(x_j, G); \theta_j, G) \\
&= \sum_{j=1}^{D} \max_{\theta_j} \sum_{i=1}^{n} \log \Pr(x_j^{(i)} \mid \text{Pa}(x_j, G); \theta_j, G)
\end{aligned}$$

So, we can select each CPT, θ_j independently from the others, given the graph structure.

The set of conditional independence assumptions is the same for any graph with the same undirected skeleton and the same set of V structures (cases where there are multiple parents of a node.)

- This makes structure search more efficient, because, when we add or remove an arc, it only requires re-estimation of the θ_j for variables X_j whose set of parents is changed from G to G'.

6.3 Bayesian score and model complexity

The maximum-likelihood score, by itself, isn't very useful: it will generally choose the model that makes no conditional independence assumptions. We can add a direct penalty term on the complexity (e.g. number of parameters) in the model. Or, we can be Bayesian!

The Bayesian score for a graphical structure is derived from

$$\Pr(G \mid \mathcal{D}) = \Pr(\mathcal{D} \mid G)\Pr(G)/\Pr(\mathcal{D}) \ ,$$

resulting in

$$\text{score}_B(G, \mathcal{D}) = \log \Pr(\mathcal{D} \mid G) + \log \Pr(G) \ .$$

We immediately see that there is an opportunity to put a prior on graph structures. That would be one way to penalize overly complex graphs. But, because of the coolness of marginal likelihood, we still get a regularization effect even when we make $\Pr(G)$ in some sense "uniform".

The *marginal likelihood* integrates out the parameter values:

$$\Pr(\mathcal{D} \mid G) = \int_{\theta_G} \Pr(\mathcal{D} \mid \theta_G, G)\Pr(\theta_G \mid G) d\theta_G \ .$$

Let's see how this works by considering two networks on two variables, X and Y:

- $G_\emptyset$: considers the two variables as being completely independent
- $G_{X\to Y}$: has X as a parent of Y

Independent model has two parameters θ_X and θ_Y.

$$\Pr(\mathcal{D} \mid G_\emptyset) = \int_{\theta_X, \theta_Y} \Pr(\mathcal{D} \mid \theta_X, \theta_Y, G_\emptyset)\Pr(\theta_X, \theta_Y \mid G_\emptyset) d\theta_X d\theta_Y \ .$$

Assuming *parameter independence* $\Pr(\theta_X, \theta_Y \mid G_\emptyset) = \Pr(\theta_X \mid G_\emptyset)\Pr(\theta_Y \mid G_\emptyset)$, we have

$$\Pr(\mathcal{D} \mid G_\emptyset) = \left(\int_{\theta_X} \Pr(\theta_X \mid G_\emptyset)\prod_{i=1}^{n} \Pr(x^{(i)} \mid \theta_X G_\emptyset) d\theta_X\right)\left(\int_{\theta_Y} \Pr(\theta_Y \mid G_\emptyset)\prod_{i=1}^{n} \Pr(y^{(i)} \mid \theta_Y, G_\emptyset) d\theta_Y\right) \ .$$

Dependent model has three parameters: θ_X, $\theta_{Y|X=0}$, and $\theta_{Y|X=1}$. Again, assuming parameter independence, we have

$$\begin{aligned}\Pr(\mathcal{D} \mid G_{X\to Y}) &= \left(\int_{\theta_X} \Pr(\theta_X \mid G_{X\to Y})\prod_{i=1}^{n} \Pr(x^{(i)} \mid \theta_X, G_{X\to Y}) d\theta_X\right) \\ &\cdot \left(\int_{\theta_{Y|X=0}} \Pr(\theta_{Y|X=0} \mid G_{X\to Y}) \prod_{\{i : x^{(i)}=0\}} \Pr(y^{(i)} \mid x^{(i)}, \theta_{Y|X=0}, G_{X\to Y}) d\theta_{Y|X=0}\right) \\ &\cdot \left(\int_{\theta_{Y|X=1}} \Pr(\theta_{Y|X=1} \mid G_{X\to Y}) \prod_{\{i : x^{(i)}=1\}} \Pr(y^{(i)} \mid x^{(i)}, \theta_{Y|X=1}, G_{X\to Y}) d\theta_{Y|X=1}\right)\end{aligned}$$

Comparison Note that the first factor is the same in both cases. In the independent model, we are using all of our data to compute a posterior in θ_Y. The more data we have, the more peaked the distribution will be and so the larger the second term will be.

In the dependent model, we are dividing our data into to groups, depending on whether $Y = 0$ or not, to estimate two parameters. We would expect the distributions on these parameters to be less peaked than the distribution on θ_Y, and therefore for the product of the second two terms in the dependent model to be smaller than the second term in the independent model.

This, even without putting a prior on model structures, using the Bayesian model selection criterion, we will tend to select more complex structures only when their improved fit of the data outweighs the higher posterior variance that results from using the same amount of data to make inferences about a higher-dimensional parameter space.

General Bayesian score with independent Dirichlet priors on each row of each CPT:

$$\Pr(\mathcal{D} \mid G) = \prod_{t=1}^{D} \left(\prod_{c=1}^{C_t} \frac{\prod_{k=1}^{K} \Gamma(N_{tck} + \alpha_{tck})}{\Gamma(\sum_{k=1}^{K} (N_{tck} + \alpha_{tck}))} \right) .$$

where $N_{tc} = \sum_k N_{tck}$, where N_{tck} is the number of times variable t has value k and its parents have value c; $\alpha_{tc} = \sum_k \alpha_{tck}$, where α_{tck} is the kth parameter to the Dirichlet prior on row c of the CPT for variable t.

This score also *decomposes* according to the graph structure, making search sort of tractable.

Picking a prior Under the assumption that we would like our prior to be *likelihood equivalent*, that is, that if G_1 and G_2 are *Markov equivalent* (have the same conditional independencies) then they have the same marginal likelihood, and we would like it to have the parameter independence property so that the score decomposes, then there is effectively only one satisfactory prior. The *BDE prior* sets

$$\alpha_{tck} = \alpha P_0(X_t = k, X_{\text{Parents}(X_t)} = c) ,$$

where α is an equivalent sample size and P_0 is some prior distribution over all possible joint configurations. If we assume that it is uniform, then we have

$$\alpha_{tck} = \alpha \frac{1}{K_t, C_t} ,$$

where K_t is the number of possible values that variable X_t can take on and C_t is the number of possible values that its parents can take on, given the graph structure.

7 Temporal models

How can we model sequential data when we don't think it is IID? If we think there is a temporal dependency, then we could try, for instance, to make a model in which the next observation depends probabilistically on the history of previous observations.

It seems very complicated to build a model in which the next output can depend on an arbitrarily long history of previous observations. We could make a model in which the most recent observation depends on a fixed number of the previous observations.

7.1 Hidden Markov models

An alternative strategy for building a fixed-size probabilistic model of arbitrary-length sequences is to postulate some (possibly hidden) *state* of the world such that, assuming that time is measured in discrete steps,

- The state at time $t+1$, Z_{t+1} is conditionally independent of the states at all previous times, $Z_1, \ldots, Z_{t-1}$, given the current state Z_t.
- The conditional probability distribution that governs the evolution of the state, $\Pr(Z_{t+1} \mid Z_t)$, is constant for all t;
- If Z_t is not observable, then we might make an observation X_t; if so, the X_t is conditionally independent of all other observations and all states at other times, given Z_t. Generally, we assume that an *observation model* $\Pr(X_t \mid Z_t)$ governs the generation of observations.

This property is called *stationarity*; it is also possible to consider non-stationary models, but we will not do so.

The joint distribution of observations and states on this model is given by:

$$\Pr(X_1, \ldots, X_n, Z_1, \ldots, Z_n) = \Pr(Z_1)\left(\prod_{t=2}^{n} \Pr(Z_t \mid Z_{t-1})\right)\left(\prod_{t=1}^{n} \Pr(X_t \mid Z_t)\right) \quad .$$

Using the d-separation criterion, there is always a path connecting any two observed variables X_i and X_j via the latent state variables, and this path is never blocked. So, the predictive distribution $\Pr(X_{t+1} \mid X_1, \ldots, X_t)$ doesn't exhibit any conditional independencies.

If the X_i are discrete, then we call this a *hidden Markov model*; if all the variables are real and the dependencies are linear-Gaussian, then it is a *linear dynamical system*. We'll focus on HMMs. They are widely used in natural language, speech, computational biology, and a variety of other fields.

HMM formal definition:

- *States* are integers $1, \ldots, K$. We will let Z_t be the state at time t, encoded as a vector of binary numbers, so if $Z_{tk} = 1$ then the system is in state k at time t.
- The *state transition model* is described by a matrix A, where

  $$A_{jk} = \Pr(z_{tk} = 1 \mid Z_{t-1,j} = 1) \quad ,$$

  Because these are probabilities, all the entries in A are in $[0, 1]$ and every row sums to 1. So

  This is called a *stochastic matrix*.

  $$\Pr(Z_t \mid Z_{t-1}) = \prod_{k=1}^{K} \prod_{j=1}^{K} A_{jk}^{z_{t-1,j} \cdot z_{t,k}} \quad .$$

  The idea is that we are using the binary encoding of the Z's to pick out the appropriate element of the transition matrix.

  This will be useful later!
- The *initial state distribution* is represented with a vector or probabilities π with elements $\pi_k = \Pr(Z_{1k} = 1)$. Then

  $$\Pr(Z_1 \mid \pi) = \prod_{k=1}^{K} \pi_k^{Z_{1k}} \quad .$$
- Observations are integers $1, \ldots, \Omega$. Let X_t be the observation at time t, encoded as a vector of binary numbers, so if $X_{tl} = 1$ then the observation at time t is l.

- The *observation model* is described by a matrix B We define

  $$B_{lk} = \Pr(X_t = l \mid Z_t = k) \ \ ,$$

  so

  $$\Pr(X_t \mid Z_t) = \prod_{k=1}^{K} \prod_{l=1}^{\Omega} B_{lk}^{z_{tk} \cdot x_{tl}} \ \ .$$

You can constrain the fundamental structure of the model by putting 0 values into the A matrix.

7.2 Exact Inference

There are specialized ways of thinking about inference problems in HMMs. Here, we will pursue an approach that takes advantage of inference methods we already understand for graphical models.

We can think of an HMM as a factor graph, with:

- A unary factor, corresponding to π, connected to Z_1:

  $$\phi_\pi(z) = \pi_z$$

- Binary factors, with values from A, between each Z_t, Z_{t+1} pair:

  $$\phi_{Z_t, Z_{t+1}}(z, w) = \Pr(Z_{t+1} = w \mid Z_t = z) \ \ .$$

- Binary factors, with values from B, between each Z_t, X_t pair:

  $$\phi_{Z_t, X_t}(z, x) = \Pr(X_t = x \mid Z_t = z) \ \ .$$

Filtering What if we are given an observation sequence $x_1, \ldots, x_n$ and want to know the current state Z_n? All we need to do is one round of message passing toward Z_n in the factor graph.

- Factor ϕ_π sends a message π to Z_1
- Each X_t, since it is observed, sends a message

  $$\mu_{X_t \to \phi_{X_t, Z_t}}(l) = x_{tl}$$

  to the factor ϕ_{X_t, Z_t}

  Recall that this is a Boolean vector with exactly one 1.

- Each factor ϕ_{X_t, Z_t} sends a message

  $$\mu_{\phi_{X_t, Z_t} \to Z_t}(k) = \sum_l x_{tl} B_{lk} \ \ ,$$

  to variable Z_t.

- Moving from left to right, first variable Z_t sends message

  $$\mu_{Z_t \to \phi_{Z_t, Z_{t+1}}}(k) \quad = \quad \mu_{\phi_{X_t, Z_t} \to Z_t}(k) \cdot \mu_{\phi_{Z_{t-1}, Z_t} \to Z_t}(k)$$

  to factor $\phi_{Z_t, Z_{t+1}}$.

- Then factor $\phi_{Z_t, Z_{t+1}}$ sends message

  $$\mu_{\phi_{Z_t, Z_{t+1}} \to Z_{t+1}}(j) = \sum_k \phi_{Z_t, Z_{t+1}}(k, j) \cdot \mu_{Z_t \to \phi_{Z_t, Z_{t+1}}}(k)$$

  to variable Z_{t+1}.

- Finally, the marginal at Z_n is just

  $$\Pr(Z_n = k) = \frac{1}{z} \mu_{\phi_{X_n, Z_n} \to Z_n}(k) \cdot \mu_{\phi_{Z_{n-1}, Z_n} \to Z_n}(k) \ \ .$$

Smoothing To figure out the marginal distribution on some other Z_s, we need to do a "backward" pass of message passing.To compute the marginals at all the Z_t,

This algorithm was developed before the general message passing sum-product algorithm, and was called, unsurprisingly, the *forward-backward* algorithm.

- Variable Z_n sends message
$$\mu_{Z_n \to \phi_{Z_{n-1},Z_n}}(k) \quad = \quad \mu_{\phi_{X_n,Z_n} \to Z_n}(k)$$
to factor ϕ_{Z_{n-1},Z_n}.
- Now, working backwards, first factor $\phi_{Z_t,Z_{t+1}}$ sends message
$$\mu_{\phi_{Z_t,Z_{t+1}} \to Z_t}(k) = \sum_j \phi_{Z_t,Z_{t+1}}(k,j) \cdot \mu_{Z_{t+1} \to \phi_{Z_t,Z_{t+1}}}(k)$$
to variable Z_t.
- Then variable Z_t sends message
$$\mu_{Z_t \to \phi_{Z_{t-1},Z_t}}(k) \quad = \quad \mu_{\phi_{X_t,Z_t} \to Z_t}(k) \cdot \mu_{\phi_{Z_t,z_{t+1}} \to z_t}(k)$$
to factor ϕ_{Z_{t-1},Z_t}.
- At any node, the conditional marginal on Z_t is
$$\Pr(Z_t = k \mid X) = \frac{1}{z}\mu_{\phi_{X_t,Z_t} \to Z_t}(k) \cdot \mu_{\phi_{Z_{t-1},Z_t} \to Z_t}(k) \cdot \mu_{\phi_{Z_t,Z_{t+1}} \to Z_t}(k) \ .$$

Pairwise marginals For learning, later, we will also need the pairwise marginals $\Pr(Z_t = j, Z_{t+1} = k \mid X)$. We can compute these easily from the messages we have already computed:

$$\begin{aligned}\Pr(Z_t = j, Z_{t+1} = k \mid X) \quad = \quad & \frac{1}{z}\mu_{\phi_{X_t,Z_t} \to Z_t}(j) \cdot \mu_{\phi_{X_{t+1},Z_{t+1}} \to Z_{t+1}}(k) \\ & \cdot \mu_{\phi_{Z_{t-1},Z_t} \to Z_t}(j) \cdot \mu_{\phi_{Z_{t+1},Z_{t+2}} \to Z_{t+1}}(k) \cdot \phi_{Z_t,Z_{t+1}}(j,k) \ .\end{aligned}$$

Probability of a sequence We might be given a sequence x and a model with parameters θ, and be asked to compute its probability $\Pr(x;\theta)$. We can compute this easily using message passing. For compactness and to remind ourselves that the transition and observation distributions are stationary, we will use ϕ_A for factors corresponding to the transition distribution matrix and ϕ_B for factors corresponding to the observation distribution matrix.

$$\begin{aligned}
\Pr(x;\theta) &= \sum_{z} \Pr(x,z;\theta) \\
&= \sum_{z_n} \Pr(x_n \mid z_n) \sum_{z_{n-1}} \Pr(z_n \mid z_{n-1}) \Pr(x_{n-1} \mid z_{n-1}) \sum_{z_{n-2}} \dots \\
&\quad \dots \sum_{z_2} \Pr(z_3 \mid z_2) \Pr(x_2 \mid z_2) \sum_{z_1} \Pr(z_2 \mid z_1) \Pr(x_1 \mid z_1) \Pr(z_1) \\
&= \sum_{z_n} \Phi_B(x_n, z_n) \sum_{z_{n-1}} \Phi_A(z_{n-1}, z_n) \Phi_B(x_{n-1}, z_{n-1}) \sum_{z_{n-2}} \dots \\
&\quad \dots \sum_{z_2} \Phi_A(z_2, z_3) \Phi_B(x_2, z_2) \sum_{z_1} \Phi_A(z_1, z_2) \phi_B(x_1, z_1) \phi_\pi(z_1) \\
&= \sum_{z_n} \Phi_B(x_n, z_n) \sum_{z_{n-1}} \Phi_A(z_{n-1}, z_n) \Phi_B(x_{n-1}, z_{n-1}) \sum_{z_{n-2}} \dots \\
&\quad \dots \sum_{z_2} \Phi_A(z_2, z_3) \Phi_B(x_2, z_2) \sum_{z_1} \Phi_A(z_1, z_2) \mu_{\phi_{X_1,Z_1} \to Z_1}(z_1) \pi(z_1) \\
&= \sum_{z_n} \Phi_B(x_n, z_n) \sum_{z_{n-1}} \Phi_A(z_{n-1}, z_n) \Phi_B(x_{n-1}, z_{n-1}) \sum_{z_{n-2}} \dots \\
&\quad \dots \sum_{z_2} \Phi_A(z_2, z_3) \Phi_B(x_2, z_2) \mu_{\phi_{Z_1,Z_2} \to Z_2}(z_2) \\
&= \sum_{z_n} \Phi_B(x_n, z_n) \sum_{z_{n-1}} \Phi_A(z_{n-1}, z_n) \Phi_B(x_{n-1}, z_{n-1}) \sum_{z_{n-2}} \dots \mu_{\phi_{Z_2,Z_3} \to Z_3}(z_3) \\
&= \sum_{z_n} \Phi_B(x_n, z_n) \sum_{z_{n-1}} \Phi_A(z_{n-1}, z_n) \Phi_B(x_{n-1}, z_{n-1}) \mu_{\phi_{Z_{n-2},Z_{n-1}} \to Z_{n-1}}(z_{n-1}) \\
&= \sum_{z_n} \Phi_B(x_n, z_n) \mu_{\phi_{Z_{n-1},Z_n} \to Z_n}(z_n)
\end{aligned}$$

So, doing a single forward pass of message passing, with the observations X fixed, and then summing out Z_n will give the probability of the sequence. *Note that this works because all of the factors are conditional probability distributions, and so the resulting probability is already normalized. If that were not true, then we would have to compute the partition function Z in order to get a calibrated probability value.*

MAP estimation: the Viterbi algorithm We might want to find the most probable *sequence* of hidden states. It is a straightforward special case of the max-product (or max-sum) algorithm.

- Pass max-sum messages to the right to find the probability of the most likely sequence; keep track, in each factor, when you maximize out a variable, which value of that variable contributed to the max
- Do a "back-tracking" pass to figure out which variable assignment at each factor contributed to the max.

7.3 Learning

Given one or more data sequences X, and letting $\theta = (A, B, \pi)$, we would like to do maximum-likelihood estimation, to find

$$\arg\max_{\theta} \Pr(X;\theta) = \arg\max_{\theta} \sum_{Z} \Pr(X,Z;\theta) \ .$$

This is difficult because:

Note that fitting a Gaussian mixture model is a related problem, in which the transition probabilities are equal to π given any state, and the observation are real valued and distributed as a Gaussian given the state.

- Summing over all possible Z is out of the question (exponential in the length of the sequence)
- The distribution doesn't factor over the X_i.

EM to the rescue! As before, we

- Start with an initial model θ^{old}
- **E step**: Given θ^{old}, we need to find both individual marginals on the hidden variables

  $$\gamma_t = \Pr(Z_t \mid X; \theta^{old})$$

  and pairwise marginals

  $$\xi_t = \Pr(Z_{t-1}, Z_t \mid X; \theta^{old}) \ .$$

  We will write $\gamma_{tk} = \Pr(z_{tk} = 1; \theta^{old})$ and $\xi_{tjk} = \Pr(z_{t-1,j} = 1, z_{tk} = 1; \theta^{old})$. We can do this efficiently with forward/backward (sum-product) as shown in the previous section.
- **M step**: We need to find

  $$\theta^{new} = \arg\max_{\theta} \sum_{k=1}^{K} \gamma_{1k} \log \pi_k + \sum_{t=2}^{n} \sum_{j=1}^{K} \sum_{k=1}^{K} \xi_{tjk} \log A_{jk} + \sum_{j=1}^{n} \sum_{k=1}^{K} \sum_{l=1}^{\Omega} \gamma_{tk} x_{tl} \log B_{lk} \ .$$

  Solve a slightly hairy Lagrange multiplier problem to get

  $$\begin{aligned} \pi_k &= \frac{\gamma_{1k}}{\sum_{j=1}^{K} \gamma_{1j}} \\ A_{jk} &= \frac{\sum_{t=2}^{n} \xi_{tjk}}{\sum_{l=1}^{K} \sum_{t=2}^{n} \xi_{tjl}} \\ B_{lk} &= \frac{\sum_{t=1}^{n} \gamma_{tk} x_{tl}}{\sum_{t=1}^{N} \gamma_{tk}} \end{aligned}$$

What do we know about this EM?

- Sensitive to initial values: any 0 value in any of the initial parameters can never be changed to a non-zero value.
- Goes to a local optimum.
- Can be modified easily to incorporate multiple sequences.
- Easily extended to any observation distribution you want.
- Scaling: be careful about loss of precision; normally need to do this all in log space, and use the log-sum-exp trick when necessary.