CHAPTER 15

Mean Field Inference

Graphical models are important and useful, but come with a serious practical problem. For many models, we cannot compute either the normalizing constant or the maximum a posteriori state. It will help to have some notation. Write X for a set of observed values, $H_1, \ldots, H_N$ for the unknown (hidden) values of interest. We will assume that these are discrete. We seek the values of $H_1, \ldots, H_N$ that maximizes $P(H_1, \ldots, H_N|X)$. There is an exponential number of such possible values, so we must exploit some kind of structure in the problem to find the maximum. In the case of a model that could be drawn as a forest, this structure was easily found; for models which can't, mostly that structure isn't there. This means the model is formally intractable — there is no practical prospect of an efficient algorithm for finding the maximum.

There are two reasons not to use this problem as a reason to simply ignore graphical models. First, graphical models that quite naturally describe interesting application problems are intractable. This chapter will work with one such model for denoising images. Second, there are quite good approximation procedures for extracting information from intractable models. This chapter will describe one such procedure.

15.1 USEFUL BUT INTRACTABLE MODELS

Here is a formal model we can use. A **Boltzmann machine** is a distribution model for a set of binary random variables. Assume we have N binary random variables U_i, which take the values 1 or -1. The values of these random variables are not observed (the true values of the pixels). These binary random variables are not independent. Instead, we will assume that some (but not all) pairs are coupled. We could draw this situation as a graph (Figure 15.1), where each node represents a U_i and each edge represents a coupling. The edges are weighted, so the coupling strengths vary from edge to edge.

Write $\mathcal{N}(i)$ for the set of random variables whose values are coupled to that of i – these are the neighbors of i in the graph. The joint probability model is

$$\log P(U|\theta) = \left[\sum_i \sum_{j \in \mathcal{N}(i)} \theta_{ij} U_i U_j \right] - \log Z(\theta) = -E(U|\theta) - \log Z(\theta).$$

Now $U_i U_j$ is 1 when U_i and U_j agree, and -1 otherwise (this is why we chose U_i to take values 1 or -1). The θ_{ij} are the edge weights; notice if $\theta_{ij} > 0$, the model generally prefers U_i and U_j to agree (as in, it will assign higher probability to states where they agree, unless other variables intervene), and if $\theta_{ij} < 0$, the model prefers they disagree.

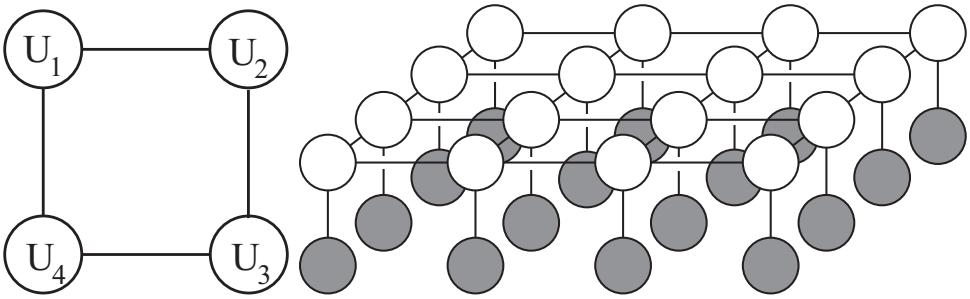

FIGURE 15.1: *On the* **left**, *a simple Boltzmann machine. Each U_i has two possible states, so the whole thing has 16 states. Different choices of the constants coupling the U's along each edge lead to different probability distributions. On the* **right**, *this Boltzmann machine adapted to denoising binary images. The shaded nodes represent the known pixel values (X_i in the text) and the open nodes represent the (unknown, and to be inferred) true pixel values H_i. Notice that pixels depend on their neighbors in the grid. There are 2^{16} states for X in this simple example.*

Here $E(U|\theta)$ is sometimes referred to as the **energy** (notice the sign - higher energy corresponds to lower probability) and $Z(\theta)$ ensures that the model normalizes to 1, so that

$$Z(\theta) = \sum_{\text{all values of U}} \left[\exp\left(-E(U|\theta)\right)\right].$$

15.1.1 Denoising Binary Images with Boltzmann Machines

Here is a simple model for a binary image that has been corrupted by noise. At each pixel, we observe the corrupted value, which is binary. Hidden from us are the true values of each pixel. The observed value at each pixel is random, but depends only on the true value. This means that, for example, the value at a pixel can change, but the noise doesn't cause blocks of pixels to, say, shift left. This is a fairly good model for many kinds of transmission noise, scanning noise, and so on. The true value at each pixel is affected by the true value at each of its neighbors – a reasonable model, as image pixels tend to agree with their neighbors.

We can apply a Boltzmann machine. We split the U into two groups. One group represents the observed value at each pixel (I will use X_i, and the convention that i chooses the pixel), and the other represents the hidden value at each pixel (I will use H_i). Each observation is either 1 or -1. We arrange the graph so that the edges between the H_i form a grid, and there is a link between each X_i and its corresponding H_i (but no other - see Figure 15.1).

Assume we know good values for θ. We have

$$P(H|X,\theta) = \frac{\exp(-E(H,X|\theta))/Z(\theta)}{\Sigma_H \left[\exp(-E(H,X|\theta))/Z(\theta)\right]} = \frac{\exp\left(-E(H,X|\theta)\right)}{\Sigma_H \exp\left(-E(H,X|\theta)\right)}$$

so posterior inference doesn't require evaluating the normalizing constant. This isn't really good news. Posterior inference still requires a sum over an exponential

number of values. Unless the underlying graph is special (a tree or a forest) or very small, posterior inference is intractable.

You might think that focusing on MAP inference will solve this problem. Recall that MAP inference seeks the values of H to maximize $P(H|X,\theta)$ or equivalently, maximizing the log of this function. We seek

$$\underset{H}{\text{argmax}} \ \log P(H|X,\theta) = (-E(H,X|\theta)) - \log\left[\Sigma_H \exp\left(-E(H,X|\theta)\right)\right]$$

but the second term is not a function of H, so we could avoid the intractable sum. This doesn't mean the problem is tractable. Some pencil and paper work will establish that there is some set of constants a_{ij} and b_j so that the solution is obtained by solving

$$\underset{H}{\text{argmax}} \ \left(\sum_{ij} a_{ij} h_i h_j\right) + \sum_j b_j h_j \quad \text{subject to } h_i \in \{-1, 1\}.$$

This is a combinatorial optimization problem with considerable potential for unpleasantness. How nasty it is depends on some details of the a_{ij}, but with the right choice of weights a_{ij}, the problem is **max-cut**, which is NP-hard.

> **Remember this:** *A natural model for denoising a binary image is to assume that there are unknown, true pixel values that tend to agree with the observed noisy pixel values and with one another. This model is intractable – you can't compute the normalizing constant, and you can't find the best set of true pixel values.*

15.1.2 A Discrete Markov Random Field

Boltzmann machines are a simple version of a much more complex device widely used in computer vision and other applications. In a Boltzmann machine, we took a graph and associated a binary random variable with each node and a coupling weight with each edge. This produced a probability distribution. We obtain a **Markov random field** by placing a random variable (doesn't have to be binary, or even discrete) at each node, and a coupling function (almost anything works) at each edge. Write U_i for the random variable at the i'th node, and $\theta(U_i, U_j)$ for the coupling function associated with the edge from i to j (the arguments tell you which function; you can have different functions on different edges).

We will ignore the possibility that the random variables are continuous. A **discrete Markov random field** has all U_i discrete random variables with a finite set of possible values. Write U_i for the random variable at each node, and $\theta(U_i, U_j)$ for the coupling function associated with the edge from i to j (the arguments tell

you which function; you can have different functions on different edges). For a discrete Markov random field, we have

$$\log P(U|\theta) = \left[\sum_i \sum_{j \in \mathcal{N}(i)} \theta(U_i, U_j)\right] - \log Z(\theta).$$

It is usual – and a good idea – to think about the random variables as indicator functions, rather than values. So, for example, if there were three possible values at node i, we represent U_i with a 3D vector containing one indicator function for each value. One of the components must be one, and the other two must be zero. Vectors like this are sometimes know as **one-hot vectors**. The advantage of this representation is that it helps keep track of the fact that the *values* that each random variable can take are not really to the point; it's the *interaction* between assignments that matters. Another advantage is that we can easily keep track of the parameters that matter. I will adopt this convention in what follows.

I will write $\mathbf{u}_i$ for the random variable at location i represented as a vector. All but one of the components of this vector are zero, and the remaining component is 1. If there are $\#(U_i)$ possible values for U_i and $\#(U_j)$ possible values for U_j, we can represent $\theta(U_i, U_j)$ as a $\#(U_i) \times \#(U_j)$ table of values. I will write $\Theta^{(ij)}$ for the table representing $\theta(U_i, U_j)$, and $\theta^{(ij)}_{mn}$ for the m, n'th entry of that table. This entry is the value of $\theta(U_i, U_j)$ when U_i takes its m'th value and U_j takes its n'th value. I write $\Theta^{(ij)}$ for a matrix whose m, n'th component is $\theta^{(ij)}_{mn}$. In this notation, I write

$$\theta(U_i, U_j) = \mathbf{u}_i^T \Theta^{(ij)} \mathbf{u}_j.$$

All this does not simplify computation of the normalizing constant. We have

$$Z(\theta) = \sum_{\text{all values of } \mathbf{u}} \left[\exp\left(\sum_i \sum_{j \in \mathcal{N}(i)} \mathbf{u}_i^T \Theta^{(ij)} \mathbf{u}_j\right)\right].$$

Note that the collection of all values of $\mathbf{u}$ has rather nasty structure, and is very big – it consists of all possible one-hot vectors representing each U.

15.1.3 Denoising and Segmenting with Discrete MRF's

A simple denoising model for images that aren't binary is just like the binary denoising model. We now use a discrete MRF. We split the U into two groups, H and X. We observe a noisy image (the X values) and we wish to reconstruct the true pixel values (the H). For example, if we are dealing with grey level images with 256 different possible grey values at each pixel, then each H has 256 possible values. The graph is a grid for the H and one link from an X to the corresponding H (like Figure 15.1). Now we think about $P(H|X, \theta)$. As you would expect, the model is intractable – the normalizing constant can't be computed.

Worked example 15.1 *A simple discrete MRF for image denoising.*

Set up an MRF for grey level image denoising.

Solution: Construct a graph that is a grid. The grid represents the true value of each pixel, which we expect to be unknown. Now add an extra node for each grid element, and connect that node to the grid element. These nodes represent the observed value at each pixel. As before, we will separate the variables U into two sets, X for observed values and H for hidden values (Figure 15.1). In most grey level images, pixels take one of 256 ($= 2^8$) values. For the moment, we work with a grey level image, so each variable takes one of 256 values. There is no reason to believe that any one pixel behaves differently from any other pixel, so we expect the $\theta(H_i, H_j)$ not to depend on the pixel location; there'll be one copy of the same function at each grid edge. By far the most usual case has

$$\theta(H_i, H_j) = \begin{bmatrix} 0 & \text{if } H_i = H_j \\ c & \text{otherwise} \end{bmatrix}$$

where $c > 0$. Representing this function using one-hot vectors is straightforward. There is no reason to believe that the relationship between observed and hidden values depends on the pixel location. However, large differences between observed and hidden values should be more expensive than small differences. Write X_j for the observed value at node j, where j is the observed value node corresponding to H_i. We usually have

$$\theta(H_i, X_j) = (H_i - X_j)^2.$$

If we think of H_i as an indicator function, then this function can be represented as a vector of values; one of these values is picked out by the indicator. Notice there is a different vector at each H_i node (because there may be a different X_i at each).

Now write $\mathbf{h}_i$ for the hidden variable at location i represented as a vector, etc. Remember, all but one of the components of this vector are zero, and the remaining component is 1. The one-hot vector representing an observed value at location i is $\mathbf{x}_i$. I write $\Theta^{(ij)}$ for a matrix who's m, n'th component is $\theta^{(ij)}_{mn}$. In this notation, I write

$$\theta(H_i, H_j) = \mathbf{h}_i^T \Theta^{(ij)} \mathbf{h}_j$$

and

$$\theta(H_i, X_j) = \mathbf{h}_i^T \Theta^{(ij)} \mathbf{x}_j = \mathbf{h}_i^T \beta_i.$$

In turn, we have

$$\log p(H|X) = \left[\left(\sum_{ij} \mathbf{h}_i^T \Theta^{(ij)} \mathbf{h}_j \right) + \sum_i \mathbf{h}_i^T \beta_i \right] + \log Z.$$

Worked example 15.2 *Denoising MRF - II*

Write out $\Theta^{(ij)}$ for the $\theta(H_i, H_j)$ with the form given in example 15.1 using the one-hot vector notation.

Solution: This is more a check you have the notation. $c\mathcal{I}$ is the answer.

Worked example 15.3 *Denoising MRF - III*

Assume that we have $X_1 = 128$ and $\theta(H_i, X_j) = (H_i - X_j)^2$. What is β_1 using the one-hot vector notation? Assume pixels take values in the range $[0, 255]$.

Solution: Again, a check you have the notation. We have

$$\beta_1 = \begin{pmatrix} 128^2 & \text{first component} \\ \dots & \\ (i-128)^2 & i\text{'th component} \\ \dots & \\ 127^2 & \end{pmatrix}$$

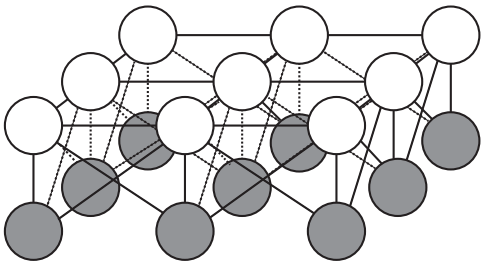

FIGURE 15.2: *For problems like image segmentation, hidden labels may be linked to many observed labels. So, for example, the segment label at one pixel might depend on the values of many pixels. This is a sketch of such a graph. The shaded nodes represent the known pixel values (X_i in the text) and the open nodes represent the (unknown, and to be inferred) labels H_i. A particular hidden node may depend on many pixels, because we will use all these pixel values to compute the cost of labelling that node in a particular way.*

Segmentation is another application that fits this recipe. We now want to break the image into a set of regions. Each region will have a label (eg "grass", "sky", "tree", etc.). The X_i are the observed values of each pixel value, and the H_i are the labels. In this case, the graph may have quite complex structure (eg

figure 15.2). We must come up with a process that computes the cost of labelling a given pixel location in the image with a given label. Notice this process could look at many other pixel values in the image to come up with the label, but not at other labels. There are many possibilities. For example, we could build a logistic regression classifier that predicts the label at a pixel from image features around that pixel (if you don't know any image feature constructions, assume we use the pixel color; if you do, you can use anything that pleases you). We then model the cost of a having a particular label at a particular point as the negative log probability of the label under that model. We obtain the $\theta(H_i, H_j)$ by assuming that labels on neighboring pixels should agree with one another, as in the case of denoising.

15.1.4 MAP Inference in Discrete MRF's can be Hard

As you should suspect, focusing on MAP inference doesn't make the difficulty go away for discrete Markov random fields.

Worked example 15.4 *Useful facts about MRF's.*

Show that, using the notation of the text, we have: (a) for any i, $\mathbf{1}^T\mathbf{h}_i = 1$; (b) the MAP inference problem can be expressed as a quadratic program, with linear constraints, on discrete variables.

Solution: For (a) the equation is true because exactly one entry in $\mathbf{h}_i$ is 1, the others are zero. But (b) is more interesting. MAP inference is equivalent to maximizing $\log p(H|X)$. Recall $\log Z$ does not depend on the $\mathbf{h}$. We seek

$$\max_{\mathbf{h}_1,\ldots,\mathbf{h}_N} \left[\left(\sum_{ij} \mathbf{h}_i^T \Theta^{(ij)} \mathbf{h}_j \right) + \sum_i \mathbf{h}_i^T \beta_i \right] + \log Z$$

subject to very important constraints. We must have $\mathbf{1}^T\mathbf{h}_i = 1$ for all i. Furthermore, any component of any $\mathbf{h}_i$ must be either 0 or 1. So we have a quadratic program (because the cost function is quadratic in the variables), with linear constraints, on discrete variables.

Example 15.4 is a bit alarming, because it implies (correctly) that MAP inference in MRF's can be very hard. You should remember this. Gradient descent is no use here because the idea is meaningless. You can't take a gradient with respect to discrete variables. If you have the background, it's quite easy to prove by producing (eg from example 15.4) an MRF where inference is equivalent to max-cut, which is NP hard.

Worked example 15.5 *MAP inference for MRF's is a linear program*

Show that, using the notation of the text, the MAP inference for an MRF problem can be expressed as a linear program, with linear constraints, on discrete variables.

Solution: If you have two binary variables z_i and z_j both in $\{0, 1\}$, then write $q_{ij} = z_i z_j$. We have that $q_{ij} \leq z_i$, $q_{ij} \leq z_j$, $q_{ij} \in \{0, 1\}$, and $q_{ij} \geq z_i + z_j - 1$. You should check (a) these inequalities and (b) that q_{ij} is uniquely identified by these inequalities. Now notice that each $\mathbf{h}_i$ is just a bunch of binary variables, and the quadratic term $\mathbf{h}_i^T \Theta^{(ij)} \mathbf{h}_j$ is linear in q_{ij}.

Example 15.5 is the start of an extremely rich vein of approximation mathematics, which we shall not mine. If you are of a deep mathematical bent, you can phrase everything in what follows in terms of approximate solutions of linear programs. For example, this makes it possible to identify MRF's for which MAP inference can be done in polynomial time; the family is more than just trees. We won't go there.

Remember this: *A natural model for denoising general images follows the line of the binary image model. One assumes that there are unknown, true pixel values that tend to agree with the observed noisy pixel values and with one another. This model is intractable – you can't compute the normalizing constant, and you can't find the best set of true pixel values. This is also a natural model of image segmentation, where the unknown values are segment identities.*

15.2 VARIATIONAL INFERENCE

We could just ignore intractable models, and stick to tractable models. This isn't a good idea, because intractable models are often quite natural. The discrete Markov random field model of an image is a fairly natural model. Image labels *should* depend on pixel values, and on neighboring labels. It is better to try and deal with the intractable model. One really successful strategy for doing so is to choose a tractable parametric family of probability models $Q(H; \theta)$, then adjust θ to find parameter values $\hat{\theta}$ that represent a distribution that is "close" in the right sense to $P(H|X)$. One then extracts information from $Q(H; \hat{\theta})$. This process is known as **variational inference**. What is remarkable is that (a) it is possible to find a $Q(H; \hat{\theta})$ without too much fuss and (b) information extracted from this distribution is often accurate and useful.

Remember this: *Variational inference tries to find a tractable distribution $Q(H;\hat{\theta})$ that is "close" to an intractable $P(H|X)$. One then extracts information from $Q(H;\hat{\theta})$.*

15.2.1 The KL Divergence

Assume we have two probability distributions $P(X)$ and $Q(X)$. A measure of their similarity is the **KL-divergence** (or sometimes **Kullback-Leibler divergence**) written

$$\mathbb{D}(P \parallel Q) = \int P(X) \log \frac{P(X)}{Q(X)} dX$$

(you've clearly got to be careful about zeros in P and Q here). This likely strikes you as an odd measure of similarity, because it isn't symmetric. It is not the case that $\mathbb{D}(P \parallel Q)$ is the same as $\mathbb{D}(Q \parallel P)$, which means you have to watch your P's and Q's. Furthermore, some work will demonstrate that it does not satisfy the triangle inequality, so KL divergence lacks two of the three important properties of a metric.

KL divergence has some nice properties, however. First, we have

$$\mathbb{D}(P \parallel Q) \geq 0$$

with equality only if P and Q are equal almost everywhere (i.e. except on a set of measure zero).

Remember this: *The KL divergence measures the similarity of two probability distributions. It is always non-negative, and is only zero if the two distributions are the same. However, it is not symmetric.*

Second, there is a suggestive relationship between KL divergence and maximum likelihood. Assume that X_i are IID samples from some *unknown* $P(X)$, and we wish to fit a parametric model $Q(X|\theta)$ to these samples. This is the usual situation we deal with when we fit a model. Now write $H(P)$ for the entropy of $P(X)$, defined by

$$H(P) = -\int P(X) \log P(X) dx = -\mathbb{E}_P[\log P].$$

The distribution P is unknown, and so is its entropy, but it is a constant. Now we can write

$$\mathbb{D}(P \parallel Q) = \mathbb{E}_P[\log P] - \mathbb{E}_P[\log Q]$$

Then

$$\begin{aligned}\mathcal{L}(\theta) = \sum_i \log Q(X_i|\theta) \approx \int P(X) \log Q(X|\theta) dX &= \mathbb{E}_{P(X)}[\log Q(X|\theta)] \\ &= -H(P) - \mathbb{D}(P \parallel Q)(\theta).\end{aligned}$$

Equivalently, we can write

$$\mathcal{L}(\theta) + \mathbb{D}(P \parallel Q)(\theta) = -H(P).$$

Recall P doesn't change (though it's unknown), so $H(P)$ is also constant (though unknown). This means that when $\mathcal{L}(\theta)$ goes up, $\mathbb{D}(P \parallel Q)(\theta)$ must go down. When $\mathcal{L}(\theta)$ is at a maximum, $\mathbb{D}(P \parallel Q)(\theta)$ must be at a minimum. All this means that, when you choose θ to maximize the likelihood of some dataset given θ for a parametric family of models, you are choosing the model in that family with smallest KL divergence from the (unknown) $P(X)$.

Remember this: *Maximum likelihood estimation recovers the parameters of a distribution in the chosen family that is closest in KL divergence to the data distribution.*

15.2.2 The Variational Free Energy

We have a $P(H|X)$ that is hard to work with (usually because we can't evaluate $P(X)$) and we want to obtain a $Q(H)$ that is "close to" $P(H|X)$. A good choice of "close to" is to require that

$$\mathbb{D}(Q(H) \parallel P(H|X))$$

is small. Expand the expression for KL divergence, to get

$$\begin{aligned}\mathbb{D}(Q(H) \parallel P(H|X)) &= \mathbb{E}_Q[\log Q] - \mathbb{E}_Q[\log P(H|X)] \\ &= \mathbb{E}_Q[\log Q] - \mathbb{E}_Q[\log P(H, X)] + \mathbb{E}_Q[\log P(X)] \\ &= \mathbb{E}_Q[\log Q] - \mathbb{E}_Q[\log P(H, X)] + \log P(X)\end{aligned}$$

which at first glance may look unpromising, because we can't evaluate $P(X)$. But $\log P(X)$ is fixed (although unknown). Now rearrange to get

$$\begin{aligned}\log P(X) &= \mathbb{D}(Q(H) \parallel P(H|X)) - (\mathbb{E}_Q[\log Q] - \mathbb{E}_Q[\log P(H, X)]) \\ &= \mathbb{D}(Q(H) \parallel P(H|X)) - \mathsf{E}_Q.\end{aligned}$$

Here

$$\mathsf{E}_Q = (\mathbb{E}_Q[\log Q] - \mathbb{E}_Q[\log P(H, X)])$$

is referred to as the **variational free energy**. We can't evaluate $\mathbb{D}(Q(H) \parallel P(H|X))$. But, because $\log P(X)$ is fixed, when E_Q goes down, $\mathbb{D}(Q(H) \parallel P(H|X))$ must

go down too. Furthermore, a minimum of E_Q will correspond to a minimum of $\mathbb{D}(Q(H) \| P(H|X))$. And we can evaluate E_Q.

We now have a strategy for building approximate $Q(H)$. We choose a family of approximating distributions. From that family, we obtain the $Q(H)$ that minimises E_Q (which will take some work). The result is the $Q(H)$ in the family that minimizes $\mathbb{D}(Q(H) \| P(H|X))$. We use that $Q(H)$ as our approximation to $P(H|X)$, and extract whatever information we want from $Q(H)$.

Remember this: *The variational free energy of Q gives a bound on the KL divergence $\mathbb{D}(Q(H) \| P(H|X))$, and is tractable if Q was chosen sensibly. We select the element of the family of Q with the smallest variational free energy.*

15.3 EXAMPLE: VARIATIONAL INFERENCE FOR BOLTZMANN MACHINES

We want to construct a $Q(H)$ that approximates the posterior for a Boltzmann machine. We will choose $Q(H)$ to have one factor for each hidden variable, so $Q(H) = q_1(H_1)q_2(H_2)\dots q_N(H_N)$. We will then assume that all but one of the terms in Q are known, and adjust the remaining term. We will sweep through the terms doing this until nothing changes.

The i'th factor in Q is a probability distribution over the two possible values of H_i, which are 1 and -1. There is only one possible choice of distribution. Each q_i has one parameter $\pi_i = P(\{H_i = 1\})$. We have

$$q_i(H_i) = (\pi_i)^{\frac{(1+H_i)}{2}} (1 - \pi_i)^{\frac{(1-H_i)}{2}}.$$

Notice the trick; the power each term is raised to is either 1 or 0, and I have used this trick as a switch to turn on or off each term, depending on whether H_i is 1 or -1. So $q_i(1) = \pi_i$ and $q_i(-1) = (1 - \pi_i)$. This is a standard, and quite useful, trick. We wish to minimize the variational free energy, which is

$$\mathsf{E}_Q = (\mathbb{E}_Q[\log Q] - \mathbb{E}_Q[\log P(H, X)]).$$

We look at the $\mathbb{E}_Q[\log Q]$ term first. We have

$$\begin{aligned} \mathbb{E}_Q[\log Q] &= \mathbb{E}_{q_1(H_1)\dots q_N(H_N)}[\log q_1(H_1) + \dots \log q_N(H_N)] \\ &= \mathbb{E}_{q_1(H_1)}[\log q_1(H_1)] + \dots \mathbb{E}_{q_N(H_N)}[\log q_N(H_N)] \end{aligned}$$

where we get the second step by noticing that

$$\mathbb{E}_{q_1(H_1)\dots q_N(H_N)}[\log q_1(H_1)] = \mathbb{E}_{q_1(H_1)}[\log q_1(H_1)]$$

(write out the expectations and check this if you're uncertain).

Now we need to deal with $\mathbb{E}_Q[\log P(H,X)]$. We have

$$\begin{aligned}\log p(H,X) &= -E(H,X) - \log Z \\ &= \sum_{i\in H}\sum_{j\in\mathcal{N}(i)\cap H}\theta_{ij}H_iH_j + \sum_{i\in H}\sum_{j\in\mathcal{N}(i)\cap X}\theta_{ij}H_iX_j + K\end{aligned}$$

(where K doesn't depend on any H and is so of no interest). Assume all the q's are known except the i'th term. Write $Q_{\hat{i}}$ for the distribution obtained by omitting q_i from the product, so $Q_{\hat{1}} = q_2(H_2)q_3(H_3)\dots q_N(H_N)$, etc. Notice that

$$\mathbb{E}_Q[\log P(H,X)] = \begin{pmatrix} q_i(-1)\mathbb{E}_{Q_{\hat{i}}}[\log P(H_1,\dots,H_i=-1,\dots,H_N,X)] + \\ q_i(1)\mathbb{E}_{Q_{\hat{i}}}[\log P(H_1,\dots,H_i=1,\dots,H_N,X)] \end{pmatrix}.$$

This means that if we fix all the q terms *except* $q_i(H_i)$, we must choose q_i to minimize

$$\begin{aligned} q_i(-1)\log q_i(-1) + q_i(1)\log q_i(1) \quad & - \\ q_i(-1)\mathbb{E}_{Q_{\hat{i}}}[\log P(H_1,\dots,H_i=-1,\dots,H_N,X)] \quad & + \\ q_i(1)\mathbb{E}_{Q_{\hat{i}}}[\log P(H_1,\dots,H_i=1,\dots,H_N,X)] & \end{aligned}$$

subject to the constraint that $q_i(1) + q_i(-1) = 1$. Introduce a Lagrange multiplier to deal with the constraint, differentiate and set to zero, and get

$$\begin{aligned} q_i(1) &= \frac{1}{c}\exp\left(\mathbb{E}_{Q_{\hat{i}}}[\log P(H_1,\dots,H_i=1,\dots,H_N,X)]\right) \\ q_i(-1) &= \frac{1}{c}\exp\left(\mathbb{E}_{Q_{\hat{i}}}[\log P(H_1,\dots,H_i=-1,\dots,H_N,X)]\right) \\ \text{where } c &= \exp\left(\mathbb{E}_{Q_{\hat{i}}}[\log P(H_1,\dots,H_i=-1,\dots,H_N,X)]\right) + \\ &\quad \exp\left(\mathbb{E}_{Q_{\hat{i}}}[\log P(H_1,\dots,H_i=1,\dots,H_N,X)]\right). \end{aligned}$$

In turn, this means we need to know $\mathbb{E}_{Q_{\hat{i}}}[\log P(H_1,\dots,H_i=-1,\dots,H_N,X)]$, etc. only up to a constant. Equivalently, we need to compute only $\log q_i(H_i) + K$ for K some unknown constant (because $q_i(1) + q_i(-1) = 1$). Now we compute

$$\mathbb{E}_{Q_{\hat{i}}}[\log P(H_1,\dots,H_i=-1,\dots,H_N,X)].$$

This is equal to

$$\mathbb{E}_{Q_{\hat{i}}}\left[\sum_{j\in\mathcal{N}(i)\cap H}\theta_{ij}(-1)H_j + \sum_{j\in\mathcal{N}(i)\cap X}\theta_{ij}(-1)X_j + \text{terms not containing } H_i\right]$$

which is the same as

$$\sum_{j\in\mathcal{N}(i)\cap H}\theta_{ij}(-1)\mathbb{E}_{Q_{\hat{i}}}[H_j] + \sum_{j\in\mathcal{N}(i)\cap X}\theta_{ij}(-1)X_j + K$$

and this is the same as

$$\sum_{j\in\mathcal{N}(i)\cap H}\theta_{ij}(-1)((\pi_j)(1) + (1-\pi_j)(-1)) + \sum_{j\in\mathcal{N}(i)\cap X}\theta_{ij}(-1)X_j + K$$

and this is

$$\sum_{j \in \mathcal{N}(i) \cap H} \theta_{ij}(-1)(2\pi_j - 1) + \sum_{j \in \mathcal{N}(i) \cap X} \theta_{ij}(-1)X_j + K.$$

If you thrash through the case for

$$\mathbb{E}_{Q_i}[\log P(H_1, \ldots, H_i = 1, \ldots, H_N, X)]$$

(which works the same) you will get

$$\begin{aligned} \log q_i(1) &= \mathbb{E}_{Q_i}[\log P(H_1, \ldots, H_i = 1, \ldots, H_N, X)] + K \\ &= \sum_{j \in \mathcal{N}(i) \cap H} [\theta_{ij}(2\pi_j - 1)] + \sum_{j \in \mathcal{N}(i) \cap X} [\theta_{ij} X_j] + K \end{aligned}$$

and

$$\begin{aligned} \log q_i(-1) &= \mathbb{E}_{Q_i}[\log P(H_1, \ldots, H_i = -1, \ldots, H_N, X)] + K \\ &= \sum_{j \in \mathcal{N}(i) \cap H} [-\theta_{ij}(2\pi_j - 1)] + \sum_{j \in \mathcal{N}(i) \cap X} [-\theta_{ij} X_j] + K \end{aligned}$$

All this means that

$$\pi_i = \frac{e^a}{e^a + e^b}$$

where

$$\begin{aligned} a &= e^{\left(\sum_{j \in \mathcal{N}(i) \cap H} [\theta_{ij}(2\pi_j - 1)] + \sum_{j \in \mathcal{N}(i) \cap X} [\theta_{ij} X_j]\right)} \\ b &= e^{\left(\sum_{j \in \mathcal{N}(i) \cap H} [-\theta_{ij}(2\pi_j - 1)] + \sum_{j \in \mathcal{N}(i) \cap X} [-\theta_{ij} X_j]\right)} \end{aligned}$$

After this blizzard of calculation, our inference algorithm is straightforward. We visit each hidden node in turn, set the associated π_i to the value of the expression above *assuming all the other π_j are fixed at their current values*, and repeat until convergence. We can test convergence by checking the size of the change in each π_j.

We can now do anything to $Q(H)$ that we would have done to $P(H|X)$. For example, we might compute the values of H that maximize $Q(H)$ for MAP inference. It is wise to limit ones ambition here, because $Q(H)$ is an approximation. It's straightforward to set up and describe, but it isn't particularly good. The main problem is that the variational distribution is unimodal. Furthermore, we chose a variational distribution by assuming that each H_i was independent of all others. This means that computing, say, covariances will likely lead to the wrong numbers (although it's easy — almost all are zero, and the remainder are easy). Obtaining an approximation by assuming that H_i is independent of all others is often called a **mean field method**.

Remember this: *A long, but easy, derivation yields a way to recover an approximation to the best denoised binary image. The inference algorithm is a straightforward optimization procedure.*

15.4 YOU SHOULD

15.4.1 remember these terms:

15.4.2 remember these facts:

15.4.3 be able to:

- Set up and solve a mean field model for denoising binary images.