University of Toronto Technical Report PSI-2003-22, April, 2003.
To appear in *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

A Comparison of Algorithms for Inference and Learning in Probabilistic Graphical Models

Brendan J. Frey and Nebojsa Jojic

Abstract

Research into methods for reasoning under uncertainty is currently one of the most exciting areas of artificial intelligence, largely because it has recently become possible to record, store and process large amounts of data. While impressive achievements have been made in pattern classification problems such as handwritten character recognition, face detection, speaker identification and prediction of gene function, it is even more exciting that researchers are on the verge of introducing systems that can perform large-scale combinatorial analyzes of data, decomposing the data into interacting components. For example, computational methods for automatic scene analysis are now emerging in the computer vision community. These methods decompose an input image into its constituent objects, lighting conditions, motion patterns, and so on. Two of the main challenges are finding effective representations and models in specific applications, and finding efficient algorithms for inference and learning in these models. In this paper, we advocate the use of graph-based probability models and their associated inference and learning algorithms. We review exact techniques and various approximate, computationally efficient techniques, including iterative conditional modes, the expectation maximization (EM) algorithm, Gibbs sampling, the mean field method, variational techniques, structured variational techniques and the sum-product algorithm and "loopy" belief propagation. We describe how each technique can be applied in a vision model of multiple, occluding objects, and contrast the behaviors and performances of the techniques using a unifying cost function, free energy.

Keywords: Graphical models, Bayesian networks, probability models, probabilistic inference, reasoning, learning, Bayesian methods, variational techniques, sum-product algorithm, loopy belief propagation, EM algorithm, mean field, Gibbs sampling, free energy, Gibbs free energy, Bethe free energy.

1 Introduction

Using the eyeball of an ox, René Descartes demonstrated in the 17th century that the backside of the eyeball contains a 2-dimensional projection of the 3-dimensional scene. Isolated during the plague, Isaac Newton slipped a bodkin between his eyeball and socket, poked the backside of his eyeball at different locations, and saw small white and colored rings of varying intensity. These discoveries helped to formalize the problem of vision: What computational mechanism can interpret a 3-dimensional scene using 2-dimensional light intensity images as input? Historically, vision has played a key role in the development of models and computational mechanisms for sensory processing and artificial intelligence.

By the mid-19th century, there were two main theories of natural vision: the “nativist theory”, where vision is a consequence of the lower nervous system and the optics of the eye, and the “empiricist theory”, where vision is a consequence of learned models created from physical and visual experiences. Hermann von Helmholtz advocated the empiricist theory, and in particular that vision involves psychological inferences in the higher nervous system, based on learned models gained from experience. He conjectured that the brain learns a generative model of how scene components are put together to explain the visual input and that vision is inference in these models [7]. A computational approach to probabilistic inference was pioneered by Thomas Bayes and Pierre-Simon Laplace in the 18th century, but it was not until the 20th century that these approaches could be used to process large amounts of data using computers. The availability of computer power motivated researchers to tackle larger problems and develop more efficient algorithms. In the past 15 years, we have seen a flurry of intense, exciting, and productive research in complex, large-scale probability models and algorithms for probabilistic inference and learning.

This paper has two purposes: First, to advocate the use of graph-based probability models for analyzing sensory input; and second, to describe and compare the latest inference and learning algorithms. Throughout the review paper, we use an illustrative example of a model that learns to describe pictures of scenes as a composition of images of foreground and background objects, selected from a learned library. We describe the latest advances in inference and learning algorithms, using the above model as a case study, and compare the behaviors and performances of the various methods. This material is based on tutorials we have run at several conferences, including *CVPR00*, *ICASSP01*, *CVPR03* and *ISIT04*.

2 Graphical Probability Models and Reasoning Under Uncertainty

In practice, our inference algorithms must cope with uncertainties in the data, uncertainties about which features are most useful for processing the data, uncertainties in the relationships between variables, and uncertainties in the value of the action that is taken as a consequence of inference. Probability theory offers a mathematically consistent way to formulate inference algorithms when reasoning under uncertainty.

There are two types of probability model. A *discriminative model* predicts the distribution of the output given the input: $P(output|input)$. Examples include linear regression, where the output is a linear function of the input, plus Gaussian noise; and SVMs where the binary class variable is Bernoulli distributed with a probability given by the distance from the input to the support vectors. A *generative model* accounts for all of the data: $P(data)$, or $P(input, output)$. An example is the factor analyzer, where the combined input/output vector is a linear function of a short, Gaussian hidden vector, plus independent Gaussian noise. Generative models can be used for discrimination by computing $P(output|input)$ using marginalization and Bayes rule. In the case of factor analysis, it turns out that the output is a linear function of a *low-dimensional representation* of the input, plus Gaussian noise.

Ng and Jordan [34] show that within the context of logistic regression, for a given problem complexity ,generative approaches work better than discriminative approaches when the training data is limited. Discriminative approaches work best when the data is extensively preprocessed, so that the amount of data relative to the complexity of the task is increased. Such preprocessing involves analyzing the unprocessed inputs that will be encountered *in situ*. This task is performed by a user who may or may not use automatic data analysis tools, and involves building a model of the input, $P(input)$, that is either conceptual or operational. An operational model can be used to perform preprocessing automatically. For example, PCA can be used to reduce the dimensionality of the input data, in the hope that the low-dimensional representation will work better for discrimination. Once an operational model of the input is available, the combination of the preprocessing model $P(input)$ and the discriminative model $P(output|input)$ corresponds to a particular decomposition of a generative model: $P(output, input) = P(output|input)P(input)$.

Generative models provide a more general way to combine the preprocessing task and the discriminative task. By jointly modeling the input and output, a generative model can discover useful, compact representations and use these to better model the data. For example, factor analysis jointly finds a low-dimensional representation that models the input *and* is good at predicting the output. In contrast, preprocessing the input using PCA, ignores the output. Also, by accounting for all of the data, a generative model can help solve one problem (*e.g.*, face detection) by solving another, related problem (*e.g.*, identifying a foreground obstruction that can explain why only part of a face is visible).

Formally, a generative model is a probability model for which the observed data is an event in the sample space. So, sampling from the model generates a sample of possible observed data. If the training data has high probability, the model is “a good fit”. However, the goal is not to find the model that is the best fit, but to find a model that fits the data well *and* is consistent with prior knowledge. Graphical

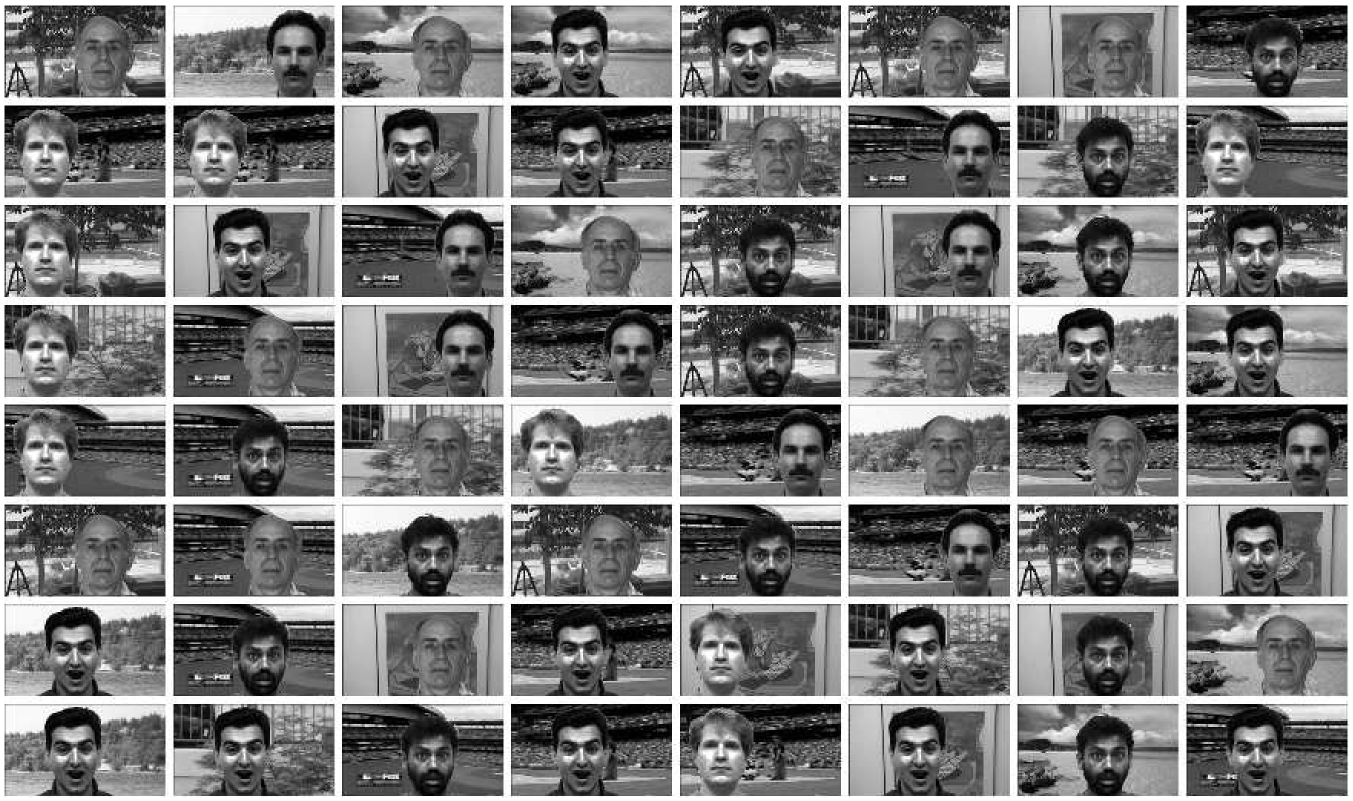


Figure 1: Some of the 300 images used to train the model in Sec. 2.1. Each image was created by randomly selecting 1 of 7 backgrounds and 1 of 5 foreground objects from the Yale face database, combining them into a 2-layer image, and adding normal noise with std. dev. of 2% of the dynamic range. Each foreground object always appears in the same location in the image, but different foreground objects appear in different places so that each pixel in the background is seen in several training images.

models provide a way to specify prior knowledge, and in particular structural prior knowledge, *e.g.*, in a video sequence, the future is independent of the past, given the current state.

2.1 Example: A Model of Foregrounds, Backgrounds and Transparency

The use of probability models in vision applications is, of course, extensive. Here, we introduce a model that is simple enough to study in detail here, but also correctly accounts for an important effect in vision: occlusion. Fig. 1 illustrates the training data. The goal of the model is to separate the 5 foreground objects and the 7 background scenes in these images. This is an important problem in vision that has broad applicability. For example, by identifying which pixels belong to the background, it is possible to improve the performance of a foreground object classifier, since errors made by noise in the background will be avoided.

The occlusion model explains an input image, with pixel intensities $z_1, \ldots, z_K$, as a composition of a foreground image and a background image (c.f. [1]), and each of these images is selected from a library of J possible images (a mixture model). Although separate libraries can be used for the foreground and background, for notational simplicity, we assume they share a common image library. The generative process is illustrated in Fig. 2a. To begin with, a foreground image is randomly selected from the library by choosing the class index f from the distribution, $P(f)$. Then, depending on the class of the foreground, a binary mask $m = (m_1, \ldots, m_K)$, $m_i \in \{0, 1\}$ is randomly chosen. $m_i = 1$ indicates that pixel z_i is a foreground pixel, whereas $m_i = 0$ indicates that pixel z_i is a background pixel. The distribution over mask RVs depends on the foreground class, since the mask must "cut out" the foreground object. However, given the foreground class, the mask RVs are chosen independently: $P(m|f) = \prod_{i=1}^{K} P(m_i|f)$. Next, the class of the background, $b \in \{1, \ldots, J\}$, is randomly chosen from $P(b)$. Finally, the intensity of the pixels in the image are selected independently, given the mask, the class of the foreground, and the class of the background: $P(z|m, f, b) = \prod_{i=1}^{K} P(z_i|m_i, f, b)$. The joint distribution is given by the following product of distributions:

$$P(z, m, f, b) = P(b)P(f)\Big(\prod_{i=1}^{K} P(m_i|f)\Big)\Big(\prod_{i=1}^{K} P(z_i|m_i, f, b)\Big). \tag{1}$$

In this equation $P(z_i|m_i, f, b)$ can be further factorized by noticing that if $m_i = 0$ the class is given by the RV b, and if $m_i = 1$ the class is given by the RV f. So, we can write $P(z_i|m_i, f, b) = P(z_i|f)^{m_i} P(z_i|b)^{1-m_i}$, where $P(z_i|f)$ and $P(z_i|b)$ are the distributions over the ith pixel intensity given by the foreground and background respectively. These distributions account for the dependence of the pixel intensity on the mixture index, as well as independent observation noise. The joint distribution can thus be written:

$$P(z, m, f, b) = P(b)P(f)\Big(\prod_{i=1}^{K} P(m_i|f)\Big)\Big(\prod_{i=1}^{K} P(z_i|f)^{m_i}\Big)\Big(\prod_{i=1}^{K} P(z_i|b)^{1-m_i}\Big). \tag{2}$$

In comparison with (1), this factorization reduces the number of arguments in some of the factors.

For representational and computational efficiency, it is often useful to specify a model using parametric distributions. Given a foreground or background class index k, we assume z_i is equal to μ_{ki} plus zero-mean Gaussian noise with variance ψ_{ki}. This noise accounts for distortions that are not explicitly modeled, such as sensor noise and fluctuations in illumination. If a Gaussian model of these noise sources is too inaccurate, extra hidden RVs can be added to better model the noise, as described in Sec. 3. Note that in the above parameterization, the foreground and background images are selected from the same library[1]. Denote the probability of class k by π_k, and let the probability that $m_i = 1$ given that the foreground class

[1]If it is desirable that the foreground and background images come from separate libraries, the class RVs f and b can be constrained, *e.g.*, so that $f \in \{1, \ldots, n\}$, $b \in \{n+1, \ldots, n+l\}$, in which case the first n images in the library are foreground images and the next l images are background images.

is f, be α_{fi}. Since the probability that $m_i = 0$ is $1 - \alpha_{fi}$, we have $P(m_i|f) = \alpha_{fi}^{m_i}(1-\alpha_{fi})^{1-m_i}$. Using these parametric forms, the joint distribution is

$$P(z, m, f, b) = \pi_b \pi_f \Big(\prod_{i=1}^{K} \alpha_{fi}^{m_i}(1-\alpha_{fi})^{1-m_i} \mathcal{N}(z_i; \mu_{fi}, \psi_{fi})^{m_i} \mathcal{N}(z_i; \mu_{bi}, \psi_{bi})^{1-m_i} \Big). \tag{3}$$

where $\mathcal{N}(z; \mu, \psi)$ is the normal density function on z with mean μ and variance ψ. An equivalent form is $P(z, m, f, b) = \pi_b \pi_f (\prod_{i=1}^{K} \alpha_{fi}^{m_i}(1-\alpha_{fi})^{1-m_i} \mathcal{N}(z_i; m_i\mu_{fi} + (1-m_i)\mu_{bi}, m_i\psi_{fi} + (1-m_i)\psi_{bi}))$, where here the mask RVs "screen" the mean and variance of the Gaussians.

In the remainder of this review paper, the above occlusion model is used as an example. One of the appeals of generative models is in their modularity and the ease with which they can be extended to cope with more complex data. In Sec. 3, we describe extensions of the occlusion model that enable it to account for motion, object deformations and object-specific changes in illumination.

2.2 Graphical Models

Graphical models describe the topology (in the sense of dependencies) of the components of a complex probability model, clarify assumptions about the representation, and lead to algorithms that make use of the topology to achieve exponential speed-ups. When constructing a complex probability model, we are faced with the following challenges: Ensuring that the model reflects our prior knowledge; Deriving efficient algorithms for inference and learning; Translating the model to a different form; Communicating the model to other researchers and users. Graphical models overcome these challenges in a wide variety of situations. After commenting on each of these issues, we briefly review 3 kinds of graphical model: Bayesian networks (BNs), Markov random fields (MRFs), and factor graphs (FGs). For a more extensive treatment, see [4, 9, 25, 26, 35].

Prior knowledge usually includes strong beliefs about the existence of hidden random variables (RVs) and the relationships between RVs in the system. This notion of "modularity" is a central aspect of graphical models. In a graphical model, the existence of a relationship is depicted by a path that connects the two RVs. Probabilistic inference in a probability model can, in principle, be carried out using Bayes rule. However, for the complex probability models that accurately describe a visual scene, direct application of Bayes rule leads to an intractable number of computations. A graphical models identifies the modules in the system and can be used to derive algorithms that make use of this modular structure to achieve exponential speedups, compared to direct application of Bayes rule. In a complex probability model, computational inference and interpretation usually benefit from judiciously groupings of RVs and these clusters should take into account dependencies between RVs. Other types of useful transformation include splitting RVs, eliminating (integrating over) RVs, and conditioning on RVs. By examining the graph, we can often easily identify transformations steps that will lead to simpler models or models that are better suited to our goals and in particular our choice of inference algorithm. For example, we may be able to transform a graphical model that contains cycles to a tree, and thus use an exact, but efficient, inference algorithm. By examining a picture of the graph, a researcher or user can quickly identify the dependency relationships between RVs in the system and understand how the influence of an RV flows through the system to change the distributions over other RVs. Whereas block diagrams enable us to efficiently communicate how computations and signals flow through a system, graphical models enable us to efficiently communicate the dependencies between components in a modular system.

2.3 Bayesian Network (BN) for the Occlusion Model

A *Bayesian network* (BN) [4,26,35] for RVs $x_1, \ldots, x_N$ is a directed acyclic graph (no directed cycles) on the set of RVs, along with one conditional probability function for each RV given its parents, $P(x_i|x_{A_i})$, where A_i is the set of indices of x_i's parents. The joint distribution is given by the product of all the conditional probability functions: $P(x) = \prod_{i=1}^{N} P(x_i|x_{A_i})$.

Fig. 2b shows the BN for the occlusion model in (1), with $K = 3$ pixels. By grouping the mask RVs together and the pixels together, we obtain the BN shown in Fig. 2c. Here, z is a real vector, $z = (z_1, z_2, z_3)$ and m is a binary vector, $m = (m_1, m_2, m_3)$. Although this graph is simpler than the graph in Fig. 2b, it is also less explicit about conditional independencies among pixels and mask RVs.

The graph indicates conditional independencies, as described in [35]. In a BN, observing a child induces a dependence between its parents. Here, the BN indicates that f and b are dependent given z and m, even though they are not (observing m decouples f and b). This demonstrates that BNs are not good at indicating conditional independence. However, the BN indicates that f and b are marginally independent, demonstrating that BNs are good at indicating marginal independence.

2.4 Markov Random Field (MRF) for the Occlusion Model

A *Markov Random Field* (MRF) [4,26,35] for RVs $x_1, \ldots, x_N$ is an undirected graph on the set of RVs, along with one potential function for each maximal clique, $g_k(x_{C_k})$, where C_k is the set of indices of the RVs in the kth maximal clique. The joint distribution is given by the product of all the potential functions, divided by a normalizing constant, $\mathcal{Z}$, called the *partition function*: $P(x) = \frac{1}{\mathcal{Z}} \prod_{k=1}^{K} g_k(x_{C_k})$, where $\mathcal{Z} = \sum_{x_1,\ldots,x_N} (\prod_{k=1}^{K} g_k(x_{C_k}))$. A clique is a fully connected subgraph, and a maximal clique is a clique

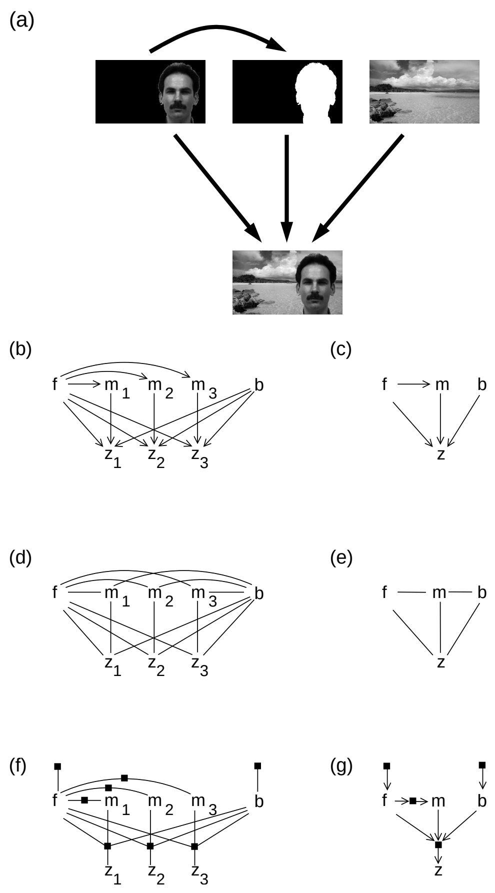


Figure 2: (a) A generative process that explains an image as a composition of the image of a foreground object with the image of the background, using a transparency map, or mask. The foreground and background are each selected stochastically from a library, and the generation of the mask depends on the foreground that was selected. We refer to this model as the *occlusion model*. (b) A BN for an occlusion model with 3 pixels, where f is the index of the foreground image, b is the index of the background image, m_i is a binary mask RV that specifies whether pixel z_i is from the foreground image ($m_i = 1$) or the background image ($m_i = 0$). (c) A simpler, but less explicit, BN is obtained by grouping the mask RVs together and the pixels together. (d) An MRF for the occlusion model. (e) An MRF corresponding to the BN in (c). (f) An FG for the occlusion model. (g) A *directed* FG expressing all properties of the BN in (c) *and* the MRF in (e).

that cannot be made larger while still being a clique. For brevity, we use the term "clique" to refer to a maximal clique, *e.g.*, the potentials on maximal cliques are usually called *clique potentials*.

The above factorization of the joint distribution is similar to the factorization for the BN, where each conditional probability function can be viewed as a clique potential. However, there is an important difference: In a BN, the conditional probability functions are individually normalized w.r.t. the child, so the product of conditional probabilities is automatically normalized, and $\mathcal{Z} = 1$.

An MRF for the occlusion model is shown in Fig. 2d and the version where the mask RVs are grouped and the pixels are grouped is shown in Fig. 2e. Note that the MRF includes an edge from m to b, indicating they are dependent, even though they are not. This demonstrates that MRFs are not good at indicating marginal independence. However, the MRF indicates f and b are independent given z and m, demonstrating that MRFs are good at indicating conditional independence.

2.5 Factor Graph (FG) for the Occlusion Model

Factor graphs (FGs) [9, 25] subsume BNs and MRFs. Any BN or MRF can be easily converted to an FG, without loss of information. Further, there exists models that have independence relationships that cannot be expressed in a BN or an MRF, but that can be expressed in an FG. Also, belief propagation takes on a simple form in FGs, so that inference in both BNs and MRFs can be simplified to a single, unified inference algorithm.

A *factor graph* (FG) for RVs $x_1, \ldots, x_N$ and *local functions* $g_1(x_{C_1}), \ldots, g_K(x_{C_K})$, is a bipartite graph on the set of RVs and a set of nodes corresponding to the functions, where each function node g_k is connected to the RVs in its argument x_{C_k}. The joint distribution is given by the product of all the functions: $P(x) = \frac{1}{\mathcal{Z}} \prod_{k=1}^{K} g_k(x_{C_k})$. In fact, $\mathcal{Z} = 1$ if the FG is a directed graph, as described below. Otherwise $\mathcal{Z}$ ensures the distribution is normalized. Note that the local functions may be positive potentials, as in MRFs, or conditional probability functions, as in BNs.

Fig. 2f shows an FG for the occlusion model. It is more explicit about the factorization of the distribution, than BNs and MRFs. As with BNs and MRFs, we can group variables to obtain a simpler FG. Also, we can indicate conditional distributions in an FG using directed edges, in which case $\mathcal{Z} = 1$. Fig. 2g shows such a *directed* FG for the model with variables grouped together. This FG expresses *all* properties of the BN and MRF. As described in [9], all independencies that can be expressed in BNs and MRFs can be expressed in FGs. Here, the directed FG indicates that f and b are independent (expressed by the BN but not the MRF), *and* it indicates that f and b are independent given z and m (expressed by the MRF but not the BN). Another advantage of FGs is that because they explicitly identify functions, they provide a useful graph for message-passing algorithms, such as belief propagation.

2.6 Converting Between FGs, BNs and MRFs

BNs and MRFs represent different independence properties, but FGs can represent all the properties that BNs and MRFs can represent.

A BN can be converted to an FG by “pinning” the edges arriving at each variable together and creating a function node associated with the conditional distribution. Directed edges are used to indicate the parent-child relationship, as shown in Fig. 2h. A directed FG can be converted to to a BN by “unpinning” each function node. An MRF can be converted to an FG by creating one function node for each maximal clique, connecting the function node to the variables in the maximal clique, and setting the function to the clique potential. An FG can be converted to an MRF by creating a maximal clique for each function node, and setting the clique potential to the function.

In fact, if a BN is converted to a directed FG and back again, the *same* BN is obtained. Similarly, if an MRF is converted to an FG and back again, the *same* MRF is obtained. Consequently, the rules for determining conditional independence in BNs and MRFs map losslessly to FGs, *i.e.*, FGs can express all conditional independencies that BNs and MRFs can express. The converse is not true: There are FGs that express independencies that cannot be expressed in a BN or an MRF, *e.g.*, the FG in Fig. 2g. It is also the case that multiple FGs may be converted to the same BN or MRF – a consequence of the fact that FGs are more explicit about factorization.

Another way to interconvert between representations is to expand the graph to include extra edges and extra variables (c.f. [38]).

3 Building Complex Models Using Modularity

Graphical models provide a way to link simpler models together in a principled fashion that respects the rules of probability theory. Fig. 3 shows how the occlusion model can be used as a module in a larger model that accounts for changing object positions, deformations, object occlusion, and changes in illumination. The figure shows a BN, where the appearance and mask vector RVs are shown as images, and the brightness, deformation and position RVs are shown pictorially. After inference and learning, the video frame is automatically decomposed into the parts shown in the BN. In previous papers, we describe efficient techniques for inference and learning in models that account for changes in object locations [11]; changes in appearances of moving objects using a subspace model [10]; common motion patterns [22]; spatial deformations in object appearance [23]; layered models of occluding objects [20]; subspace models of occluding objects [12]; and the “epitome” of components in object appearance and shape [21]. An inference and learning algorithm in a combined model, like the one shown above, can be obtained by linking together the modules and associated algorithms.

4 Parameterized Models and the Exponential Family

So far, we have studied graphical models as representations of structured probability models for computer vision. We now turn to the general problem of how to learn these models from training data. For the purpose of learning, it is often convenient to express the conditional distributions or potentials in a graphical model as parameterized functions. Choosing the forms of the parameterized functions usually restricts the model class, but can make computations easier. For example, Sec. 2.1 shows how we can parameterize the conditional probability functions in the occlusion model.

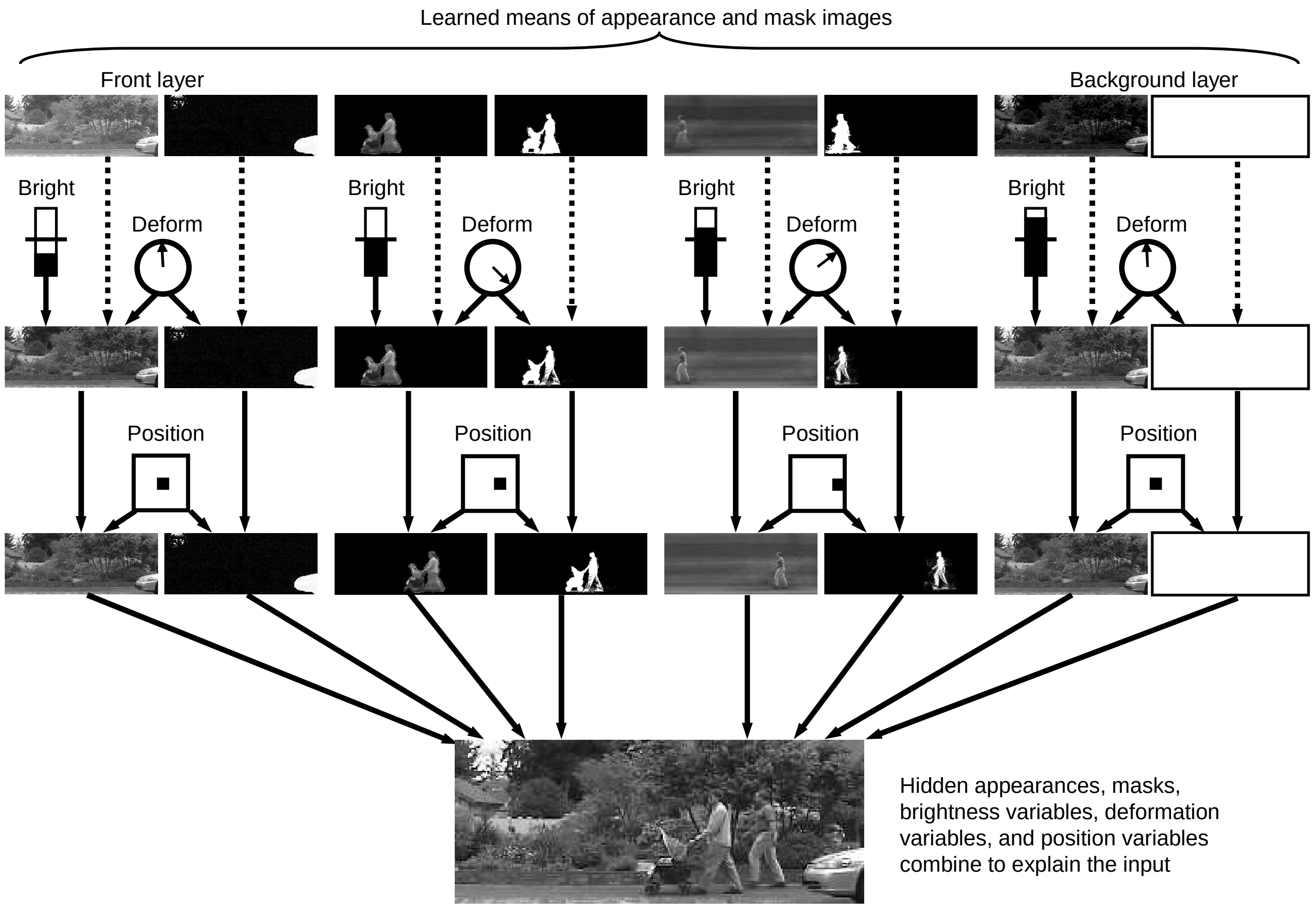

Learned means of appearance and mask images
Front layer
Background layer
Bright
Deform
Bright
Deform
Bright
Deform
Bright
Deform
Position
Position
Position
Position
Hidden appearances, masks, brightness variables, deformation variables, and position variables combine to explain the input

Figure 3: Simple probability models can be combined in a principled way to build a more complex model that can be learned from training data. Here, after the model parameters (some shown in the top row of pictures) are learned from the input video, the model explains a particular video frame as a composition of 4 "card-board cutouts", each of which is decomposed into appearance, transparency (mask), position, brightness and deformation (which accounts for the gait of the walking person).

4.1 Parameters as RVs

Usually, the model parameters are not known exactly, but we have prior knowledge and experimental results that provide evidence for plausible values of the model parameters. Interpreting the parameters as RVs, we can include them in the conditional distributions or potentials that specify the graphical model, and encode our prior knowledge in the form of a distribution over the parameters.

Including the parameters as RVs in the occlusion model, we obtain the following conditional distributions: $P(b|\pi) = \pi_b$, $P(f|\pi) = \pi_f$, $P(m_i|f, \alpha_{1i}, \ldots, \alpha_{Ji}) = \alpha_{fi}^{m_i}(1-\alpha_{fi})^{1-m_i}$, $P^f(z_i|f, \mu_{1i}, \ldots, \mu_{Ji}, \psi_{1i}, \ldots, \psi_{Ji})$ $= \mathcal{N}(z_i; \mu_{fi}, \psi_{fi})$, $P^b(z_i|b, \mu_{1i}, \ldots, \mu_{Ji}, \psi_{1i}, \ldots, \psi_{Ji}) = \mathcal{N}(z_i; \mu_{bi}, \psi_{bi})$. We obtain a simpler model (but one that is less specific about independencies) by clustering the mask RVs, the pixels, the mask parameters, and the pixel means and variances. The resulting conditional distributions are $P(b|\pi) = \pi_b$, $P(f|\pi) = \pi_f$, $P(m|f, \alpha) = \prod_{i=1}^{K} \alpha_{fi}^{m_i}(1 - \alpha_{fi})^{1-m_i}$, $P(z|m, f, b, \mu, \psi, \mu, \psi) = \prod_{i=1}^{K} \mathcal{N}(z_i; \mu_{fi}, \psi_{fi})^{m_i}$

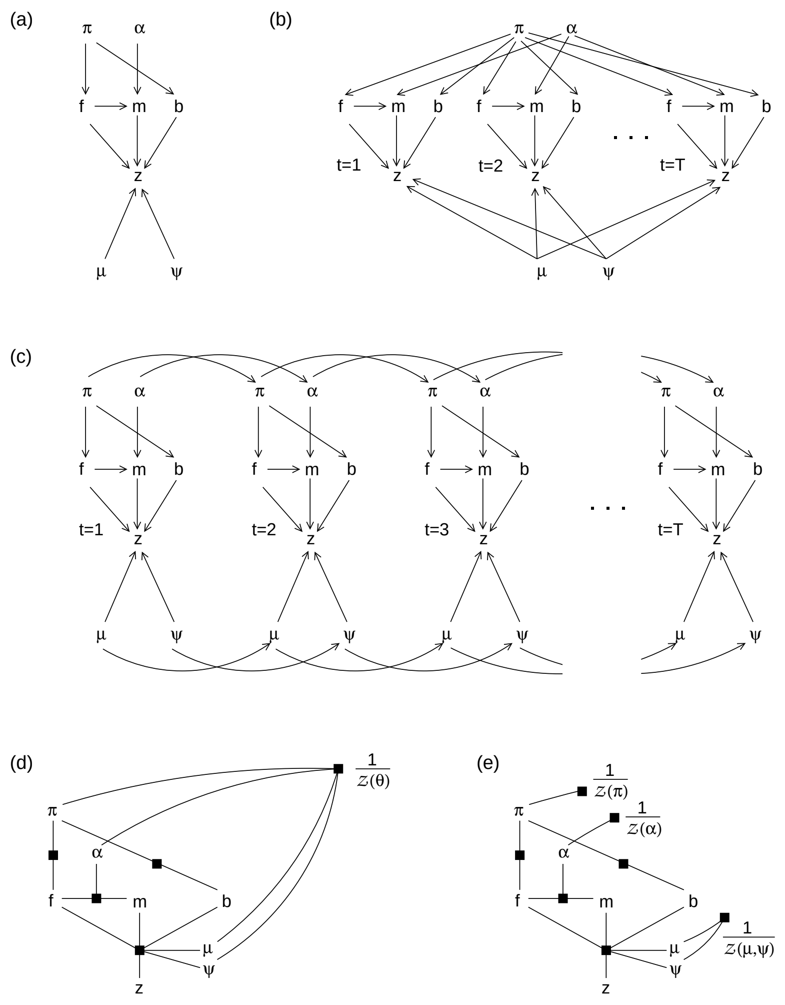


Figure 4: (a) The parameter sets π, α, μ and ψ can be included in the BN as RVs. (b) For a training set with T i.i.d. cases, these parameters are shared across all training cases. (c) If the training cases are time-series data (*e.g.* a video sequence), we may create one parameter set for each time instance, but require the parameters to change slowly over time. (d) Generally, undirected graphical models must include a normalization function $1/\mathcal{Z}(\theta)$, which makes inference of the parameters more difficult. Viewing the occlusion model as a member of the exponential family, we can draw an undirected FG, which includes the function, $1/\mathcal{Z}(\theta)$. (e) When the parameters specify conditional distributions, $\mathcal{Z}(\theta)$ factorizes into local terms, leading to a representation that is equivalent to the one in (a).

$\mathcal{N}(z_i; \mu_{bi}, \psi_{bi})^{1-m_i}$.

Since we are interpreting the parameters as RVs, we must specify a distribution for them. Generally, the distribution over parameters can be quite complex, but simplifying assumptions can be made for the sake of computational expediency, as describe in later sections. For now, we assume that $P(\pi, \alpha, \mu, \psi, \mu) = P(\pi)P(\alpha)P(\mu)P(\psi)$. The BN for this parameterized model is shown in Fig. 4a, and the joint distribution over RVs and parameters is $P(z, m, f, b, \pi, \alpha, \mu, \psi) = P(b|\pi)P(f|\pi)P(m|f, \alpha)P(z|m, f, b, \mu, \psi)P(\pi)P(\alpha)$ $P(\mu)P(\psi)$.

4.2 Introducing Training Data

Training data can be used to infer plausible configurations of the model parameters. We imagine that there is a setting of the parameters that produced the training data. However, since we only see the training data, there will be many settings of the parameters that are good matches to the training data, so the best we can do is compute a distribution over the parameters.

Denote the hidden RVs by h and the visible RVs by v. The hidden RVs can be divided into the parameters, denoted by θ, and one set of hidden RVs $h^{(t)}$, for each of the training cases, $t = 1, \ldots, T$. So, $h = (\theta, h^{(1)}, \ldots, h^{(T)})$. Similarly, there is one set of visible RVs for each training case: $v = (v^{(1)}, \ldots, v^{(T)})$. Assuming the training cases are independent and identically drawn (i.i.d.), the distribution over all visible RVs and hidden RVs (including parameters) is

$$P(h, v) = P(\theta) \prod_{t=1}^{T} P(h^{(t)}, v^{(t)} | \theta).$$

$P(\theta)$ is the parameter prior and $\prod_{t=1}^{T} P(h^{(t)}, v^{(t)}|\theta)$ is the likelihood. In the occlusion model described above, we have $\theta = (\mu, \psi, \pi, \alpha)$, $h^{(t)} = (f^{(t)}, b^{(t)}, m^{(t)})$, and $v^{(t)} = z^{(t)}$. The BN for T i.i.d. training cases is shown in Fig. 4b.

When the training cases consist of time-series data (such as a video sequence), the parameters often can be thought of as RVs that change slowly over time. Fig. 4c shows the above model, where there is a different set of parameters for each training case, but where we assume the parameters are coupled across time. Using $^{(t)}$ to denote the training case at time $t = 1, \ldots, T$, the following distributions couple the parameters across time: $P(\pi^{(t)}|\pi^{(t-1)})$, $P(\alpha^{(t)}|\alpha^{(t-1)})$, $P(\mu^{(t)}|\mu^{(t-1)})$, $P(\psi^{(t)}|\psi^{(t-1)})$. The uncertainty in these distributions specifies how quickly the parameters can change over time. Such a model can be viewed as the basis for on-line learning algorithms. For simplicity, in this paper, we assume the model parameters are fixed for the entire training set.

4.3 The Exponential Family

Members of the *exponential family* [2] have the following parameterization: $P(x|\theta) = (1/\mathcal{Z}(\theta)) \exp\big(\sum_i \theta_i \Omega_i(x)\big)$, where $\theta = (\theta_1, \theta_2, \ldots)$ is a parameter vector and $\Omega_i(x)$ is the ith *sufficient statistic*. The sufficient statistics of x contain all information that is needed to determine the density of x. $\mathcal{Z}(\theta)$ is the partition function, which normalizes $P(x|\theta)$: $\mathcal{Z}(\theta) = \sum_x \exp(\sum_i \theta_i \Omega_i(x))$. For members of the exponential family, there is a simple relationship between the distribution for one training case and the distribution for an entire training set. Let $x^{(t)}$ be the hidden *and* visible variables for the tth training case.

Then $P(x^{(t)}|\theta) = \exp(\sum_i \theta_i \Omega_i(x^{(t)}))/\mathcal{Z}(\theta)$ and the likelihood for the entire training set is $P(x|\theta) = \prod_t P(x^{(t)}|\theta) = \exp(\sum_i \theta_i(\sum_{t=1}^T \Omega_i(x^{(t)})))/\mathcal{Z}(\theta)^T$. The sufficient statistics for the entire training set are given by summing the sufficient statistics over training cases.

To put the occlusion model in exponential family form, note that the sufficient statistics for a normal density on z_i are z_i and z_i^2. The reader can confirm that the joint distribution can be written $P(z, m, f, b) = (1/\mathcal{Z}(\theta)) \exp\big(\sum_{j=1}^J (\ln \pi_j)\{[b = j]\} + \sum_{j=1}^J (\ln \pi_j)\{[f = j]\} + \sum_{i=1}^K \sum_{j=1}^J (\ln \alpha_{ji})\{[m_i = 1][f = j]\} + \sum_{i=1}^K \sum_{j=1}^J (\ln(1 - \alpha_{ji}))\{[m_i = 0][f = j]\} - \sum_{i=1}^K \sum_{j=1}^J (1/2\psi_{ji})\{z_i^2[m_i = 1][f = j]\} + \sum_{i=1}^K \sum_{j=1}^J (\mu_{ji}/\psi_{ji})\{z_i[m_i = 1][f = j]\} - \sum_{i=1}^K \sum_{j=1}^J (1/2\psi_{ji})\{z_i^2[m_i = 0][b = j]\} + \sum_{i=1}^K \sum_{j=1}^J (\mu_{ji}/\psi_{ji}) \{z_i[m_i = 0][b = j]\}\big)$, where curly braces identify the sufficient statistics, and square braces indicate Iverson's notation: $[expr] = 1$ if $expr$ is true, and $[expr] = 0$ if $expr$ is false.

Modular structure in members of the exponential family arises when each sufficient statistic $\Omega_i(x)$ depends on a subset of RVs x_{C_i} with indices C_i. Then, $P(x) = (1/\mathcal{Z}(\theta)) \prod_i \exp\big(\theta_i \Omega_i(x_{C_i})\big)$, so we can express $P(x)$ using a graphical model, *e.g.* a FG. In the FG, there can be one function node for each sufficient statistic Ω_i and one variable node for each parameter θ_i, but a more succinct FG is obtained by grouping related sufficient statistics together and grouping their corresponding parameters together. Fig. 4d shows a FG for the exponential family representation of the occlusion model, where we have made groups of π's, α's, μ's and ψ's. Note that the FG must include the normalizing function $1/\mathcal{Z}(\theta)$.

Generally, computing $\mathcal{Z}(\theta)$ is intractable since we must sum or integrate over x. However, if the exponential family parameterization corresponds to a BN, the sufficient statistics can be grouped so that each group defines a conditional distribution in the BN. In this case, $\mathcal{Z}(\theta)$ simplifies to a product of local partition functions, where each local partition function ensures that the corresponding conditional distribution is normalized. In the above model, the normalization constants associated with the conditional distributions for f, m, b and z are uncoupled, so we can write $\mathcal{Z}(\theta) = \mathcal{Z}(\pi)\mathcal{Z}(\alpha)\mathcal{Z}(\mu)\mathcal{Z}(\psi)$, where, *e.g.*, $\mathcal{Z}(\psi) = \prod_{jk} \sqrt{2\pi\psi_{jk}}$. Fig. 4e shows the FG in this case, which has the same structure as the BN in Fig. 4a.

4.4 Uniform and Conjugate Parameter Priors

Parameter priors encode the cost of specific configurations of the parameters. For simplicity, the *uniform prior* is often used, where $P(\theta) = const$. Then, $P(h, v) \propto \prod_{t=1}^T P(h^{(t)}, v^{(t)}|\theta)$, and the dependence of the parameters on the data is determined solely by the likelihood. In fact, a uniform prior is not uniform under a different parameterization. Also, the uniform density for the real numbers does not exist, so the uniform prior is *improper*. However, these facts are often ignored for computational expediency. Importantly, the use of a uniform prior is justified when the amount of training data is large relative to the maximum model complexity, since then the prior will have little effect on the model. One exception is zeros in the prior, which can never be overcome by the likelihood, but such hard constraints can be incorporated in the learning algorithm, *e.g.*, using Lagrange multipliers.

Assuming a uniform prior for all parameters in the occlusion model, the joint distribution over RVs and parameters is

$$
\begin{aligned}
&P(\mu,\psi,\pi,\alpha,f^{(1)},b^{(1)},m^{(1)},\dots,f^{(T)},b^{(T)},m^{(T)},z^{(1)},\dots,z^{(T)}) \\
&\propto \prod_{t=1}^{T}\Big(\pi_{f^{(t)}}\pi_{b^{(t)}}\Big(\prod_{i=1}^{K}\alpha_{f^{(t)}i}^{m_i^{(t)}}(1-\alpha_{f^{(t)}i})^{1-m_i^{(t)}}\mathcal{N}(z_i^{(t)};\mu_{f^{(t)}i},\psi_{f^{(t)}i})^{m_i^{(t)}}\mathcal{N}(z_i^{(t)};\mu_{b^{(t)}i},\psi_{b^{(t)}i})^{1-m_i^{(t)}}\Big)\Big).
\end{aligned}
\tag{4}
$$

Note that when using uniform priors, parameter constraints, such as $\sum_{i=1}^{J}\pi_i = 1$, must be taken into account during inference and learning.

The *conjugate prior* offers the same computational advantage as the uniform prior, but allows specification of stronger prior knowledge and is also a proper prior. The idea is to choose a prior that has the same form as the likelihood, so the prior can be thought of as the likelihood of fake, user-specified data. The joint distribution over parameters and RVs is given by the likelihood of both the real data *and* the fake data. For members of the exponential family, the fake training data takes the form of extra, user-specified terms added to each sufficient statistic, *e.g.*, extra counts added for Bernoulli RVs.

In the occlusion model, imagine that before seeing the training data, we observe λ_j fake examples from image class j. The likelihood of the fake data for parameter π_j is ${\pi_j}^{\lambda_j}$, so the conjugate prior for $\pi_1,\dots,\pi_J$ is $P(\pi_1,\dots,\pi_J)\propto\prod_{j=1}^{J}{\pi_j}^{\lambda_j}$ if $\sum_{j=1}^{J}\pi_j = 1$ and 0 otherwise. This is the Dirichlet distribution and $P(\pi_1,\dots,\pi_J)$ is the Dirichlet prior. The conjugate prior for the mean of a Gaussian distribution is a Gaussian distribution, because the RV and its mean appear symmetrically in the Gaussian pdf. The conjugate prior for the *inverse variance* β of a Gaussian distribution is a Gamma distribution. Imagine fake data consisting of λ examples where the squared difference between the RV and its mean is δ^2. The likelihood for this fake data is proportional to $(\beta^{1/2}e^{-\delta^2\beta/2})^{\lambda} = \beta^{\lambda/2}e^{-(\lambda\delta^2/2)\beta}$. This is a Gamma distribution in β with mean $1/\delta^2 + 2/\lambda\delta^2$ and variance $2(1/\delta^2 + 2/\lambda\delta^2)/\lambda\delta^2$. Setting the prior for β to be proportional to this likelihood, we see that the conjugate prior for the inverse variance is the Gamma distribution.

5 Algorithms for Inference and Learning

Once a generative model describing the image rendering process has been specified, vision consists of probabilistic inference. In Fig. 4b, for training images $z^{(1)}, \ldots, z^{(T)}$, vision consists of inferring the set of mean images and variance maps, μ, ψ, the mixing proportions π, the set of binary mask probabilities, α, and, for every training case, the class of the foreground image, f, the class of the background image, b, and the binary mask used to combine these images, m.

Exact inference is often intractable, so we turn to approximate algorithms that search for distributions that are close to the correct posterior distribution. This is accomplished by minimizing pseudo-distances on distributions, called "free energies". (For an alternative view, see [36].) It is interesting that in the 1800's, Helmholtz was one of the first researchers to propose that vision is inference in a generative model, *and* that nature seeks correct probability distributions in physical systems by minimizing free energy. Although there is no record that Helmholtz saw that the brain might perform vision by minimizing a free energy, we can't help but wonder if he pondered this.

Viewing parameters as RVs, inference algorithms for RVs and parameters alike make use of the conditional independencies in the graphical model. It is possible to describe graph-based propagation algorithms for updating distributions over parameters [16]. It is often important to treat parameters and RVs differently during inference. Whereas each RV plays a role in a single training case, the parameters are shared across many training cases. So, the parameters are impacted by more evidence than RVs and are often pinned down more tightly by the data. This observation becomes relevant when we study approximate inference techniques that obtain point estimates of the parameters, such as the expectation maximization algorithm [6].

We now turn to the general problem of inferring the values of unobserved (hidden) RVs, given the values of the observed (visible) RVs. Denote the hidden RVs by h and the visible RVs by v and partition the hidden RVs into the parameters θ and one set of hidden RVs $h^{(t)}$, for each training case $t = 1, \ldots, T$. So, $h = (\theta, h^{(1)}, \ldots, h^{(T)})$. Similarly, there is one set of visible RVs for each training case: $v = (v^{(1)}, \ldots, v^{(T)})$. Assuming the training cases are i.i.d., the distribution over all RVs is

$$P(h, v) = P(\theta)\Big(\prod_{t=1}^{T} P(h^{(t)}, v^{(t)}|\theta)\Big). \tag{5}$$

In the occlusion model, $\theta = (\mu, \psi, \pi, \alpha)$, $h^{(t)} = (f^{(t)}, b^{(t)}, m^{(t)})$, and $v^{(t)} = z^{(t)}$.

Exact inference consists of computing estimates or making decisions based on the posterior distribution over all hidden RVs (including the parameters), $P(h|v)$. From Bayes rule,

$$P(h|v) = \frac{P(h,v)}{\int_h P(h,v)},$$

where we use the notation $\int_h$ to include *summing* over discrete hidden RVs. The denominator normalizes the distribution, but if we need only a proportional function, $P(h,v)$ suffices, since w.r.t. h, $P(h|v) \propto P(h,v)$. In the case of a graphical model, $P(h,v)$ is equal to either the product of the conditional distributions, or the product of the potential functions, divided by the partition function.

5.1 Partition Functions Complicate Learning

For undirected graphical models and general members of the exponential family, $P(x,\theta) = P(\theta)\frac{1}{\mathcal{Z}(\theta)}\prod_i \phi_i(x_{C_i})$ and $\ln P(x,\theta) = \ln P(\theta) - \ln \mathcal{Z}(\theta) + \sum_i \ln \phi_i(x_{C_i})$. When adjusting a particular parameter, the sum of log-potentials nicely isolates the influence to those potentials that depend on the parameter, but the partition function makes all parameters interdependent. Generally, as shown in Fig. 4d, $\mathcal{Z}(\theta)$ induces dependencies between all parameters. Since $\mathcal{Z}(\theta) = \sum_x(\prod_i \phi_i(x_{C_i}))$, exactly determining the influence of a parameter change on the partition function is often intractable. In fact, determining this influence can also be viewed as a problem of approximate inference, since the partition function is the complete marginalization of $\prod_i \phi_i(x_{C_i})$. So, many of the techniques discussed in this paper can be used to approximately determine the effect of the partition function (*e.g.,* Gibbs sampling [19]). There are also learning techniques that are specifically aimed at undirected graphical models, such as iterative proportional fitting [4].

For directed models, the partition function factorizes into local partition functions (c.f. Fig. 4e), so the parameters can be directly inferred using the techniques described in this paper.

5.2 Model Selection

Often, some aspects of the model structure are known, but others are not. In the occlusion model, we may be confident about the structure of the BN in Fig. 4b, but not the number of classes, J. Unknown structure can be represented as a hidden RV, so that inference of this hidden RV corresponds to Bayesian model selection [17, 27]. The BN in Fig. 4b can be modified to include an RV, J, whose children are all of the f and b variables and where J limits the range of the class indices. Given a training set, the posterior over J reveals how probable the different models are. When model structure is represented in this way, proper priors should be specified for all model parameters, so that the probability density of the extra parameters needed in more complex models is properly accounted for. For an example of Bayesian learning of infinite mixture models, see [31].

5.3 Numerical Issues

Many inference algorithms rely on the computation of expressions of the form $p = \prod_j a_j^{q_j}$, where the number of terms can be quite large. To avoid underflow, it is common to work in the log-domain. Denoting the log-domain value of a variable by "$\tilde{}$", we can compute $\tilde{p} \leftarrow \sum_j q_j \tilde{a}_j$. If p is needed, set $p \leftarrow \exp(\tilde{p})$. Keep in mind that if $\tilde{p}$ is large and negative, p may be set to 0. This problem can be avoided when computing a *normalized* set of p_i's (*e.g.*, probabilities). Suppose $\tilde{p}_i$ is the log-domain value of the unnormalized version of p_i. Since the p_i's are to be normalized, we can add a constant to the $\tilde{p}_i$'s to raise them to a level where numerical underflow will not occur when taking exponentials. Computing $\tilde{m} \leftarrow \max_i \tilde{p}_i$ and then setting $\tilde{p}_i \leftarrow \tilde{p}_i - \tilde{m}$, will ensure that $\max_i \tilde{p}_i = 0$, so one of the $\exp(\tilde{p}_i)$'s will be 1. Next, compute the log-normalizing constant, $\tilde{c} \leftarrow \ln(\sum_i \exp(\tilde{p}_i))$. The previous step ensures that the sum in this expression will produce a strictly positive number, avoiding $\ln 0$. Finally, the $\tilde{p}_i$'s are normalized, $\tilde{p}_i \leftarrow \tilde{p}_i - \tilde{c}$, and, if needed, the p_i's are computed, $p_i \leftarrow \exp(\tilde{p}_i)$. In some cases, notably when computing *joint* probabilities of RVs and observations using the sum-product algorithm, we need to compute the unnormalized sum $s = \sum_i p_i$, where each p_i is so small that it is stored in its log-domain form, $\tilde{p}_i$. The above method can be used, but $\tilde{m}$ must be added back in to retain the unnormalized form. First, compute $\tilde{m} \leftarrow \max_i \tilde{p}_i$ and then set $\tilde{s} \leftarrow \tilde{m} + \ln(\sum_i \exp(\tilde{p}_i - \tilde{m}))$.

5.4 Exact Inference in the Occlusion Model

We consider two cases: Exact inference when the model parameters are known, and exact inference when the model parameters are unknown. When the model parameters are known, the distribution over the hidden RVs is given in (3). f and b each take on J values and there are K binary mask RVs, so the total number of configurations of f, b and m is $J^2 2^K$. For moderate model sizes, even if we can compute the posterior, we cannot store the posterior probability of every configuration. However, from the BN in Fig. 2b, we see that m_i is independent of $m_j, j \neq i$, given f, b and z_i (the Markov blanket of m_i). Thus, we represent the posterior distribution as follows:

$$P(m, f, b|z) = P(f, b|z)P(m|f, b, z) = P(f, b|z) \prod_{i=1}^{K} P(m_i|f, b, z).$$

Here, the posterior can be stored using $O(J^2)$ numbers[2] for $P(f, b|z)$ and for each configuration of f and b, $O(K)$ numbers for the probabilities $P(m_i|f, b, z)$, $i = 1, \ldots, K$, giving a total storage requirement of $O(KJ^2)$ numbers. Using the fact that $P(m_i|f, b, z) = P(m_i|f, b, z_i) \propto P(z_i, m_i|f, b) =$

[2]We use $O(\cdot)$ to indicate the number of scalar memory elements or binary scalar operations, up to a constant.

$P(m_i|f,b)P(z_i|m_i,f,b) = P(m_i|f)P(z_i|m_i,f,b)$ and substituting the definitions of the conditional distributions, we have

$$P(m_i = 1|f,b,z) = \frac{\alpha_{fi}\mathcal{N}(z_i;\mu_{fi},\psi_{fi})}{\alpha_{fi}\mathcal{N}(z_i;\mu_{fi},\psi_{fi}) + (1-\alpha_{fi})\mathcal{N}(z_i;\mu_{bi},\psi_{bi})}.$$

We need only store $P(m_i = 1|f,b,z)$, since $P(m_i = 0|f,b,z) = 1 - P(m_i = 1|f,b,z)$). For each $i = 1,\ldots,K$ and each configuration of f and b, this can be computed and normalized using a small number of multiply-adds. The total number of multiply-adds needed to compute $P(m_i = 1|f,b,z)$ for all i is $O(KJ^2)$.

$P(f,b|z)$ can be computed as follows:

$$\begin{aligned} P(f,b|z) &= \sum_m P(m,f,b|z) \propto \sum_m P(m,f,b,z) \\ &= \pi_b\pi_f \prod_{i=1}^{K}\Big(\sum_{m_i}\Big(\alpha_{fi}^{m_i}(1-\alpha_{fi})^{1-m_i}\mathcal{N}(z_i;\mu_{fi},\psi_{fi})^{m_i}\mathcal{N}(z_i;\mu_{bi},\psi_{bi})^{1-m_i}\Big)\Big) \\ &= \pi_b\pi_f \prod_{i=1}^{K}\Big(\alpha_{fi}\mathcal{N}(z_i;\mu_{fi},\psi_{fi}) + (1-\alpha_{fi})\mathcal{N}(z_i;\mu_{bi},\psi_{bi})\Big). \end{aligned}$$

For each value of f and b, this can be computed using $O(K)$ multiply-adds. Once it is computed for all J^2 combinations of f and b, the result is normalized to give $P(f,b|z)$. The total number of multiply-adds needed to compute $P(f,b|z)$ is $O(KJ^2)$. Combining this with the above technique, the exact posterior over f, b and m can be computed in $O(KJ^2)$ multiply-adds and stored in $O(KJ^2)$ numbers.

When the parameters are not known, we must infer the distribution over them, as well as the RVs. Assuming a uniform parameter prior, the posterior distribution over parameters and hidden RVs in the occlusion model of Fig. 4b is proportional to the joint distribution given in (4). This posterior can be thought of as a very large mixture model. There are $J^{2T}2^{KT}$ discrete configurations of the class RVs and the mask RVs and for each configuration, there is a distribution over the real-valued parameters. In each mixture component, the class probabilities are Dirichlet-distributed and the mask probabilities are Beta-distributed. (The Beta pdf is the Dirichlet pdf when there is only one free parameter.) The pixel means and variances are coupled in the posterior, but given the variances, the means are normally distributed and given the means, the inverse variances are Gamma-distributed. If the training data is processed sequentially, where one training case is absorbed at a time, the mixture posterior can be updated as shown in [5].

The exact posterior is intractable, because the number of posterior mixture components is exponential in the number of training cases, and the posterior distribution over the pixel means and variances are coupled. In the remainder of this paper, we describe a variety of approximate inference techniques and discuss advantages and disadvantages of each approach.

5.5 Approximate Inference as Minimizing Free Energies

Usually, the above techniques cannot be applied directly to $P(h|v)$, because this distribution cannot be computed in a tractable manner. So, we must turn to various approximations.

Many approximate inference techniques can be viewed as minimizing a cost function called "free energy" [33], which measures the accuracy of an approximate probability distribution. These include iterative conditional modes [3], the expectation maximization (EM) algorithm [6, 33], variational techniques [24,33] structured variational techniques [24], Gibbs sampling [32] and the sum-product algorithm (a.k.a. loopy belief propagation) [25,35].

The idea is to approximate the true posterior distribution $P(h|v)$ by a *simpler* distribution $Q(h)$, which is then used for making decisions, computing estimates, summarizing the data, *etc.* Here, approximate inference consists of searching for the distribution $Q(h)$ that is closest to $P(h|v)$. A natural choice for a measure of similarity between the two distributions is the relative entropy (a.k.a. Kullback-Leibler divergence):

$$D(Q, P) = \int_h Q(h) \ln \frac{Q(h)}{P(h|v)}.$$

This is a divergence, not a distance, because it is not symmetric: $D(Q, P) \neq D(P, Q)$. However, $D(Q, P)$ is similar to a distance in that $D(Q, P) \geq 0$, and $D(Q, P) = 0$ if and only if the approximating distribution exactly matches the true posterior, $Q(h) = P(h|v)$. The reason we use $D(Q, P)$ and not $D(P, Q)$ is that the former computes the expectation w.r.t. the simple distribution, Q, whereas the latter computes the expectation w.r.t. P, which is generally very complex[3].

Approximate inference techniques can be derived by examining ways of searching for $Q(h)$, to minimize $D(Q, P)$. In fact, directly computing $D(Q, P)$ is usually intractable, because it depends on $P(h|v)$. If we already have a tractable form for $P(h|v)$ to insert into the expression for $D(Q, P)$, we probably don't have a need for approximate inference. Fortunately, $D(Q, P)$ can be modified in a way that does not alter the structure of the search space of $Q(h)$, but makes computations tractable. If we subtract $\ln P(v)$

[3]For example, if $Q(h) = \prod_i Q(h_i)$ then $D(P, Q) = \int_h P(h|v) \ln P(h|v) - \sum_i \int_{h_i} P(h_i|v) \ln Q(h_i)$. Under the constraint $\int_{h_i} Q(h_i) = 1$, the minimum of $D(P, Q)$ is given by $Q(h_i) = P(h_i|v)$. However, computing $P(h_i|v)$ is an NP-hard problem, so minimizing $D(P, Q)$ is also an NP-hard problem.

from $D(Q, P)$, we obtain

$$F(Q, P) = D(Q, P) - \ln P(v) = \int_h Q(h) \ln \frac{Q(h)}{P(h|v)} - \int_h Q(h) \ln P(v) = \int_h Q(h) \ln \frac{Q(h)}{P(h, v)}. \tag{6}$$

Notice that $\ln P(v)$ does not depend on $Q(h)$, so subtracting $\ln P(v)$ will not influence the search for $Q(h)$. For BNs and directed FGs, we *do* have a tractable expression for $P(h, v)$, namely the product of conditional distributions.

If we interpret $-\ln P(h, v)$ as the energy function of a physical system and $Q(h)$ as a distribution over the state of the system, then $F(Q, P)$ is equal to the average energy minus the entropy. In statistical physics, this quantity is called the *free energy* of the system (a.k.a., *Gibbs free energy* or *Helmholtz free energy*). Nature tends to minimize free energies, which corresponds to finding the equilibrium Boltzmann distribution of the physical system.

Another way to derive the free energy is by using Jensen's inequality to bound the log-probability of the visible RVs. Jensen's inequality states that a concave function of a convex combination of points in a vector space is greater than or equal to the convex combination of the concave function applied to the points. To bound the log-probability of the visible RVs, $\ln P(v) = \ln(\int_h P(h, v))$, we use an arbitrary distribution $Q(h)$ (a set of convex weights) to obtain a convex combination inside the concave $\ln()$ function:

$$\ln P(v) = \ln\Big(\int_h P(h, v)\Big) = \ln\Big(\int_h Q(h) \frac{P(h, v)}{Q(h)}\Big) \geq \int_h Q(h) \ln\Big(\frac{P(h, v)}{Q(h)}\Big) = -F(Q, P).$$

We see that the free energy is an upper bound on the negative log-probability of the visible RVs: $F(Q, P) \geq -\ln P(v)$. This can also be seen by noting that $D(Q, P) \geq 0$ in (6).

Free energy for i.i.d. training cases

From (5), for a training set of T i.i.d. training cases with hidden RVs $h = (\theta, h^{(1)}, \ldots, h^{(T)})$ and visible RVs $v = (v^{(1)}, \ldots, v^{(T)})$, we have $P(h, v) = P(\theta) \prod_{t=1}^{T} P(h^{(t)}, v^{(t)}|\theta)$. The free energy is

$$F(Q, P) = \int_h Q(h) \ln Q(h) - \int_\theta Q(\theta) \ln P(\theta) - \sum_{t=1}^{T} \int_{h^{(t)}, \theta} Q(h^{(t)}, \theta) \ln P(h^{(t)}, v^{(t)}|\theta). \tag{7}$$

The decomposition of F into a sum of one term for each training case simplifies learning.

Exact inference revisited

The idea of approximate inference is to search for $Q(h)$ in a space of models that are simpler than the true posterior $P(h|v)$. It is instructive to not assume $Q(h)$ is simplified and derive the minimizer of

$F(Q,P)$. The only constraint we put on $Q(h)$ is that it is normalized: $\sum_h Q(h) = 1$. To account for this constraint, we form a Lagrangian from $F(Q,P)$ with Lagrange multiplier λ and optimize $F(Q,P)$ w.r.t. $Q(h)$: $\partial(F(Q,P) + \lambda \int_h Q(h))/\partial Q(h) = \ln Q(h) + 1 - \ln P(h,v) + \lambda$. Setting this derivative to 0 and solving for λ, we find $Q(h) = P(h,v)/\int_h P(h,v) = P(h|v)$. So, minimizing the free energy without any simplifying assumptions on $Q(h)$ produces exact inference. The minimum free energy is $\min_Q F(Q,P) = \int_h P(h|v)\ln(P(h|v)/P(h,v)) = -\ln P(v)$. The minimum free energy is equal to the negative log-probability of the data. This minimum is achieved when $Q(h) = P(h|v)$.

Revisiting exact inference in the occlusion model

In the occlusion model, if we allow the approximating distribution $Q(f,b,m)$ to be unconstrained, we find that the minimum free energy is obtained when $Q(f,b,m) = P(f,b|z)\prod_{i=1}^{K} P(m_i|f,b,z)$. Of course, nothing is gained computationally by using this Q-distribution. In the following sections, we see how the use of various approximate forms for $Q(f,b,m)$ lead to tremendous speed-ups.

5.6 MAP Estimation as Minimizing Free Energy

Maximum *a posteriori* (MAP) estimation searches for $\hat{h} = \arg\max_h P(h|v)$, which is the same as $\arg\max_h P(h,v)$. For discrete hidden RVs, MAP estimation minimizes $F(Q,P)$ using a Q-distribution of the form $Q(h) = [h = \hat{h}]$, where $[expr] = 1$ if $expr$ is true, and $[expr] = 0$ if $expr$ is false. The free energy in (6) simplifies to $F(Q,P) = \sum_h [h = \hat{h}] \ln[h = \hat{h}]/P(h,v) = -\ln P(\hat{h},v)$, *i.e.*, minimizing $F(Q,P)$ is equivalent to maximizing $P(\hat{h},v)$.

For continuous hidden RVs, the Q-distribution for a point estimate is a Dirac delta function centered at the estimate: $Q(h) = \delta(h - \hat{h})$. The free energy in (6) reduces to $F(Q,P) = \int_h \delta(h-\hat{h}) \ln \delta(h - \hat{h})/P(h,v) = -\ln P(\hat{h},v) - H_\delta$, where H_δ is the entropy of the Dirac delta. This entropy does not depend on $\hat{h}$, so minimizing $F(Q,P)$ corresponds to searching for values of $\hat{h}$ that maximize $P(\hat{h},v)$[4]. Two popular methods that use point inferences are iterative conditional modes and the EM algorithm.

5.7 Iterative Conditional Modes (ICM)

The most famous example of ICM is *k-means clustering*, where the hidden RVs are the cluster centers and the class labels. Here, ICM iterates between assigning each training case to the closest cluster center, and setting each cluster center equal to the average of the training cases assigned to it. ICM is popular

[4]In fact, $H_\delta \to -\infty$. To see this, define $\delta(x) = 1/\epsilon$ if $0 \le x \le \epsilon$ and 0 otherwise. Then, $H_\delta = \ln \epsilon$, which goes to $-\infty$ as $\epsilon \to 0$. This infinite penalty in $F(Q,P)$ is a reflection of the fact that an infinite-precision point-estimate of h does a very poor job of representing the uncertainty in h under $P(h|v)$.

because it easy to implement. However, ICM does not take into account uncertainties in hidden RVs during inference, causing it to find poor local minima.

ICM works by searching for a configuration of h that maximizes $P(h|v)$. The simplest version of ICM examines each hidden RV h_i in turn, and sets the RV to its MAP value, given all other RVs. Since all hidden RVs but h_i are kept constant in this update, only the RVs in the Markov blanket of h_i are relevant. Denote these RVs by x_{M_i} and denote the product of all conditional distributions or potentials that depend on h_i by $f(h_i, x_{M_i})$. ICM proceeds as follows:

Initialization. Pick values for all hidden RVs h (randomly, or cleverly).

ICM Step. Consider one of the hidden RVs, h_i. Holding all other RVs constant, set h_i to its MAP value:

$$h_i \leftarrow \mathrm{argmax}_{h_i} P(h_i|h \setminus h_i, v) = \mathrm{argmax}_{h_i} f(h_i, x_{M_i}).$$

where $h \setminus h_i$ is the set of all hidden RVs other than h_i.

Repeat for a fixed number of iterations or until convergence.

If h_i is discrete, this procedure is straightforward. If h_i is continuous and exact optimization of h_i is not possible, its current value can be used as the initial point for a search algorithm, such as a Newton method or a gradient-based method.

The free energy for ICM is the free energy described above, for general point inferences.

ICM in the occlusion model

Even when the model parameters in the occlusion model are known, the computational cost of exact inference can be rather high. When the number of clusters J is large, examining all J^2 configurations of the foreground class and the background class is computationally burdensome. For ICM in the occlusion model, the Q-distribution for the entire training set is $Q = (\prod_k \delta(\pi_k - \hat{\pi}_k))(\prod_{k,i} \delta(\mu_{ki} - \hat{\mu}_{ki}))(\prod_{k,i} \delta(\psi_{ki} - \hat{\psi}_{ki}))(\prod_{k,i} \delta(\alpha_{ki} - \hat{\alpha}_{ki}))(\prod_t [b^{(t)} = \hat{b}^{(t)}])(\prod_t [f^{(t)} = \hat{f}^{(t)}])(\prod_t \prod_i [m_i^{(t)} = \hat{m}_i^{(t)}])$. Substituting this Q-distribution and the P-distribution in (4) into the expression for the free energy in (7), we obtain the following:

$$\begin{aligned} F &= -\sum_t \Big(\ln \hat{\pi}_{\hat{f}^{(t)}} + \ln \hat{\pi}_{\hat{b}^{(t)}}\Big) - \sum_t \Big(\sum_i \hat{m}_i^{(t)} \ln \hat{\alpha}_{\hat{f}^{(t)}i} + (1-\hat{m}_i^{(t)}) \ln(1-\hat{\alpha}_{\hat{f}^{(t)}i})\Big) \\ &\quad + \sum_t \sum_i \hat{m}_i^{(t)} \Big((z_i^{(t)} - \hat{\mu}_{\hat{f}^{(t)}i})^2 / 2\hat{\psi}_{\hat{f}^{(t)}i} + \ln(2\pi\hat{\psi}_{\hat{f}^{(t)}i})/2\Big) \\ &\quad + \sum_t \sum_i (1-\hat{m}_i^{(t)}) \Big((z_i^{(t)} - \hat{\mu}_{\hat{b}^{(t)}i})^2 / 2\hat{\psi}_{\hat{b}^{(t)}i} + \ln(2\pi\hat{\psi}_{\hat{b}^{(t)}i})/2\Big) - H. \end{aligned}$$

H is the entropy of the δ-functions and is constant during optimization. F measures the mismatch between the input image and the image obtained by combining the foreground and background using the mask.

To minimize the free energy w.r.t. all RVs and parameters, we can iteratively solve for each RV or parameters by setting the derivative of F to 0, keeping the other RVs and parameters fixed. These updates can be applied in any order, but since the model parameters depend on values of all hidden RVs, we first optimize for all hidden RVs, and then optimize for model parameters. Furthermore, since for every observation, the class RVs depend on all pixels, when updating the hidden RVs, we first visit the mask values for all pixels and then the class RVs.

After all parameters and RVs are set to random values, the updates are applied recursively, as described in Fig. 5. To keep notation simple, the " ˆ " symbol is dropped and in the updates for the variables m_i, b and f, the training case index $^{(t)}$ is dropped.

5.8 Block ICM and Conjugate Gradients

One problem with the simple version of ICM described above is its severe greediness. Suppose $f(h_i, x_{M_i})$ has almost the same value for two different values of h_i. ICM will pick one value for h_i, discarding the fact that the other value of h_i is almost as good. This problem can be partly avoided by optimizing subsets of h, instead of single elements of h. At each step of this *block ICM* method, a tractable subgraph of the graphical model is selected, and all RVs in the subgraph are updated to maximize $P(h, v)$. Often, this can be done efficiently using the max-product algorithm [25]. An example of this method is training HMMs using the Viterbi algorithm to select the most probable state sequence. For continuous hidden RVs, an alternative to block ICM is to use a joint optimizer, such as a conjugate gradients.

5.9 The Expectation-Maximization Algorithm

The EM algorithm accounts for uncertainty in some RVs, while performing ICM-like updates for the other RVs. Typically, for parameters θ and remaining RVs $h^{(1)}, \ldots, h^{(T)}$, EM obtains point estimates for θ and computes the exact posterior over the other RVs, given θ. The Q-distribution is $Q(h) = \delta(\theta - \hat{\theta})Q(h^{(1)}, \ldots, h^{(T)})$. Recall that for i.i.d. data, $P(h, v) = P(\theta)(\prod_{t=1}^{T} P(h^{(t)}, v^{(t)}|\theta))$. Given θ, the RVs associated with different training cases are independent, so we have $Q(h) = \delta(\theta - \hat{\theta}) \prod_{t=1}^{T} Q(h^{(t)})$. In exact EM, no restrictions are placed on the distributions, $Q(h^{(t)})$.

Substituting $P(h, v)$ and $Q(h)$ into (6), we obtain the free energy:

$$F(Q, P) = -\ln P(\hat{\theta}) + \sum_{t=1}^{T} \Big(\int_{h^{(t)}} Q(h^{(t)}) \ln \frac{Q(h^{(t)})}{P(h^{(t)}, v^{(t)}|\hat{\theta})} \Big).$$

EM alternates between minimizing $F(Q,P)$ w.r.t. the set of distributions $Q(h^{(1)}),\ldots,Q(h^{(T)})$ in the E step, and minimizing $F(Q,P)$ w.r.t. $\hat{\theta}$ in the M step.

When updating $Q(h^{(t)})$, the only constraint is that $\int_{h^{(t)}} Q(h^{(t)}) = 1$. As described earlier, this constraint is accounted for using a Lagrange multiplier. Setting the derivative of $F(Q,P)$ to zero and solving for $Q(h^{(t)})$, we obtain the solution, $Q(h^{(t)}) = P(h^{(t)}|v^{(t)},\hat{\theta})$. Taking the derivative of $F(Q,P)$ w.r.t. $\hat{\theta}$, we obtain

$$\frac{\partial F(Q,P)}{\partial \hat{\theta}} = -\frac{\partial}{\partial \hat{\theta}} \ln P(\hat{\theta}) - \sum_{t=1}^{T} \Big(\int_{h^{(t)}} Q(h^{(t)}) \frac{\partial}{\partial \hat{\theta}} \ln P(h^{(t)}, v^{(t)}|\hat{\theta}) \Big).$$

For M parameters, this is a set of M equations. These two solutions give the EM algorithm:

Initialization. Choose values for the parameters, $\hat{\theta}$ (randomly, or cleverly).

E Step. Minimize $F(Q,P)$ w.r.t. Q using exact inference, by setting

$$Q(h^{(t)}) \leftarrow P(h^{(t)}|v^{(t)},\hat{\theta}),$$

for each training case, given the parameters $\hat{\theta}$ and the data $v^{(t)}$.

M Step. Minimize $F(Q,P)$ w.r.t. the model parameters $\hat{\theta}$ by solving

$$-\frac{\partial}{\partial \hat{\theta}} \ln P(\hat{\theta}) - \sum_{t=1}^{T} \Big(\int_{h^{(t)}} Q(h^{(t)}) \frac{\partial}{\partial \hat{\theta}} \ln P(h^{(t)}, v^{(t)}|\hat{\theta}) \Big) = 0. \tag{8}$$

This is the derivative of the expected log-probability of the complete data. For M parameters, this is a system of M equations. Often, the prior on the parameters is assumed to be uniform, $P(\hat{\theta}) = const$, in which case the first term in the above expression vanishes.

Repeat for a fixed number of iterations or until convergence.

In Sec. 5.5, we showed that when $Q(h) = P(h|v)$, $F(Q,P) = -\ln P(v)$. So, the EM algorithm alternates between obtaining a tight lower bound on $\ln P(v)$ and then maximizing this bound w.r.t. the model parameters. This means that with each iteration, the log-probability of the data, $\ln P(v)$, must increase or stay the same.

EM in the occlusion model

As with ICM we approximate the distribution over the parameters using $Q(\theta) = \delta(\theta - \hat{\theta})$. As described above, in the E step we set $Q(b,f,m) \leftarrow P(b,f,m|z)$ for each training case, where, as described in

ICM

E Step (Variable Updates)

```
For $t = 1, \ldots, T$:
    $f \leftarrow \mathrm{argmax}_f \Big(\pi_f \prod_{i=1}^K \Big(\alpha_{fi}^{m_i}(1-\alpha_{fi})^{1-m_i}\mathcal{N}(z_i;\mu_{fi},\psi_{fi})^{m_i}\Big)\Big)$
    For $i = 1, \ldots, K$:
        $m_i \leftarrow \mathrm{argmax}_{m_i} \left\{ \begin{array}{ll} \alpha_{fi}\mathcal{N}(z_i;\mu_{fi},\psi_{fi}) & \text{if } m_i = 1 \\ (1-\alpha_{fi})\mathcal{N}(z_i;\mu_{bi},\psi_{bi}) & \text{if } m_i = 0 \end{array} \right\}$
    $b \leftarrow \mathrm{argmax}_b \Big(\pi_b \prod_{i=1}^K \mathcal{N}(z_i;\mu_{bi},\psi_{bi})^{1-m_i}\Big)$
```

M Step (Parameter Updates)

```
For $j = 1, \ldots, J$:  $\pi_j \leftarrow \frac{1}{2T}\sum_{t=1}^T([f^{(t)} = j] + [b^{(t)} = j])$
For $j = 1, \ldots, J$:  For $i = 1, \ldots, K$:
    $\alpha_{ji} \leftarrow \frac{\sum_{t=1}^T [f^{(t)}=j] m_i^{(t)}}{\sum_{t=1}^T [f^{(t)}=j]}$
    $\mu_{ji} \leftarrow \frac{\sum_{t=1}^T [f^{(t)}=j \text{ or } b^{(t)}=j] z_i^{(t)}}{\sum_{t=1}^T [f^{(t)}=j \text{ or } b^{(t)}=j]}$
    $\psi_{ji} \leftarrow \frac{\sum_{t=1}^T [f^{(t)}=j \text{ or } b^{(t)}=j](z_i^{(t)}-\mu_{ji})^2}{\sum_{t=1}^T [f^{(t)}=j \text{ or } b^{(t)}=j]}$
```

Exact EM

E Step

```
For $t = 1, \ldots, T$:
    $Q(b,f) \leftarrow c_2\ \pi_b\pi_f \prod_{i=1}^K \Big(\alpha_{fi}\mathcal{N}(z_i;\mu_{fi},\psi_{fi}) + (1-\alpha_{fi})\mathcal{N}(z_i;\mu_{bi},\psi_{bi})\Big)$
    $Q(b) \leftarrow \sum_f Q(b,f), \quad Q(f) \leftarrow \sum_b Q(b,f)$
    For $i = 1, \ldots, K$:
        $Q(m_i = 1|b,f) \leftarrow c_1\ \alpha_{fi}\mathcal{N}(z_i;\mu_{fi},\psi_{fi})$
        $Q(m_i = 0|b,f) \leftarrow c_1\ (1-\alpha_{fi})\mathcal{N}(z_i;\mu_{bi},\psi_{bi})$
    For $i = 1, \ldots, K$:
        $Q(m_i,b) \leftarrow \sum_f Q(m_i|b,f)Q(b,f), \quad Q(m_i,f) \leftarrow \sum_b Q(m_i|b,f)Q(b,f)$
```

M Step

```
For $j = 1, \ldots, J$:  $\pi_j \leftarrow \frac{1}{2T}\sum_t (Q(f^{(t)} = j) + Q(b^{(t)} = j))$
For $j = 1, \ldots, J$:  For $i = 1, \ldots, K$:
    $\alpha_{ji} \leftarrow \frac{\sum_t Q(m_i^{(t)}=1, f^{(t)}=j)}{\sum_t Q(f^{(t)}=j)}$
    $\mu_{ji} \leftarrow \frac{\sum_t \big(Q(m_i^{(t)}=1,f^{(t)}=j)+Q(m_i^{(t)}=0,b^{(t)}=j)\big) z_i^{(t)}}{\sum_t \big(Q(m_i^{(t)}=1,f^{(t)}=j)+Q(m_i^{(t)}=0,b^{(t)}=j)\big)}$
    $\psi_{ki} \leftarrow \frac{\sum_t \big(Q(m_i^{(t)}=1,f^{(t)}=j)+Q(m_i^{(t)}=0,b^{(t)}=j)\big)(z_i^{(t)}-\mu_{ji})^2}{\sum_t \big(Q(m_i^{(t)}=1,f^{(t)}=j)+Q(m_i^{(t)}=0,b^{(t)}=j)\big)}$
```

Gibbs Sampling EM

E Step

```
For $t = 1, \ldots, T$:
    $f \leftarrow \mathrm{sample}_f \ \Big(\pi_f \prod_{i=1}^K \Big(\alpha_{fi}\mathcal{N}(z_i;\mu_{fi},\psi_{fi})\Big)^{m_i}(1-\alpha_{fi})^{1-m_i}\Big)$
    For $i = 1, \ldots, K$:
        $m_i \leftarrow \mathrm{sample}_{m_i} \left\{ \begin{array}{ll} \alpha_{fi}\mathcal{N}(z_i;\mu_{fi},\psi_{fi}) & \text{if } m_i = 1 \\ (1-\alpha_{fi})\mathcal{N}(z_i;\mu_{bi},\psi_{bi}) & \text{if } m_i = 0 \end{array} \right\}$
    $b \leftarrow \mathrm{sample}_b \ \Big(\pi_b \prod_{i=1}^K \mathcal{N}(z_i;\mu_{bi},\psi_{bi})^{1-m_i}\Big)$
```

M Step

```
For $j = 1, \ldots, J$:  $\pi_j \leftarrow \frac{1}{2T}\sum_t([f^{(t)} = j] + [b^{(t)} = j])$
For $j = 1, \ldots, J$:  For $i = 1, \ldots, K$:
    $\alpha_{ji} \leftarrow \frac{\sum_t [f^{(t)}=j] m_i^{(t)}}{\sum_t [f^{(t)}=j]}$
    $\mu_{ji} \leftarrow \frac{\sum_t [f^{(t)}=j \text{ or } b^{(t)}=j] z_i^{(t)}}{\sum_t [f^{(t)}=j \text{ or } b^{(t)}=j]}$
    $\psi_{ji} \leftarrow \frac{\sum_t [f^{(t)}=j \text{ or } b^{(t)}=j](z_i^{(t)}-\mu_{ji})^2}{\sum_t [f^{(t)}=j \text{ or } b^{(t)}=j]}$
```

Variational EM

E Step

```
For $t = 1, \ldots, T$:
    $Q(f) \leftarrow c_3\ \pi_f \prod_{i=1}^K \Big(\Big(\alpha_{fi}\mathcal{N}(z_i;\mu_{fi},\psi_{fi})\Big)^{Q(m_i=1)}(1-\alpha_{fi})^{Q(m_i=0)}\Big)$
    For $i = 1, \ldots, K$:
        $Q(m_i = 1) \leftarrow c_1\ \prod_f (\alpha_{fi}\mathcal{N}(z_i;\mu_{fi},\psi_{fi}))^{Q(f)}$
        $Q(m_i = 0) \leftarrow c_1\ (\prod_f (1-\alpha_{fi})^{Q(f)})(\prod_b \mathcal{N}(z_i;\mu_{bi},\psi_{bi})^{Q(b)})$
    $Q(b) \leftarrow c_2\ \pi_b \prod_{i=1}^K \mathcal{N}(z_i;\mu_{bi},\psi_{bi})^{Q(m_i=0)}$
```

M Step

```
For $j = 1, \ldots, J$:  $\pi_j \leftarrow \frac{1}{2T}(\sum_t Q(f^{(t)} = j) + \sum_t Q(b^{(t)} = j))$
For $j = 1, \ldots, J$:  For $i = 1, \ldots, K$:
    $\alpha_{ji} \leftarrow \frac{\sum_t Q(m_i^{(t)}=1)Q(f^{(t)}=j)}{\sum_t Q(f^{(t)}=j)}$
    $\mu_{ji} \leftarrow \frac{\sum_t \big(Q(m_i^{(t)}=1)Q(f^{(t)}=j)+Q(m_i^{(t)}=0)Q(b^{(t)}=j)\big) z_i^{(t)}}{\sum_t \big(Q(m_i^{(t)}=1)Q(f^{(t)}=j)+Q(m_i^{(t)}=0)Q(b^{(t)}=j)\big)}$
    $\psi_{ki} \leftarrow \frac{\sum_t \big(Q(m_i^{(t)}=1)Q(f^{(t)}=j)+Q(m_i^{(t)}=0)Q(b^{(t)}=j)\big)(z_i^{(t)}-\mu_{ji})^2}{\sum_t \big(Q(m_i^{(t)}=1)Q(f^{(t)}=j)+Q(m_i^{(t)}=0)Q(b^{(t)}=j)\big)}$
```

Figure 5: Inference and learning algorithms for the occlusion model. Iverson's notation is used, where $[expr] = 1$ if $expr$ is true, and $[expr] = 0$ if $expr$ is false. The constant c is used to normalize distributions.

Structured Variational EM

E Step

```
For $t = 1, \ldots, T$:
  $Q(f) \leftarrow c_1\ \pi_f \prod_{i=1}^{K} \left( \frac{\alpha_{fi}}{Q(m_i=1|f)} \mathcal{N}(z_i; \mu_{fi}, \psi_{fi}) \right)^{Q(m_i=1|f)}$
                $\cdot \left( \frac{1-\alpha_{fi}}{Q(m_i=0|f)} \prod_{b=1}^{J} \mathcal{N}(z_i; \mu_{bi}, \psi_{bi})^{Q(b)} \right)^{Q(m_i=0|f)}$
  For $i = 1, \ldots, K$:
    $Q(m_i = 1|f) \leftarrow c_2\ \alpha_{fi} \mathcal{N}(z_i; \mu_{fi}, \psi_{fi})$
    $Q(m_i = 0|f) \leftarrow c_2\ (1 - \alpha_{fi}) \prod_b \mathcal{N}(z_i; \mu_{bi}, \psi_{bi})^{Q(b)}$
    $Q(m_i, f) \leftarrow Q(m_i|f)Q(f), \quad Q(m_i) \leftarrow \sum_f Q(m_i, f)$
  $Q(b) \leftarrow c_3\ \pi_b \prod_{i=1}^{K} \mathcal{N}(z_i; \mu_{bi}, \psi_{bi})^{Q(m_i=0)}$
```

M Step

```
For $j = 1, \ldots, J$: $\pi_j \leftarrow \frac{1}{2T} (\sum_t Q(f^{(t)} = j) + \sum_t Q(b^{(t)} = j))$
For $j = 1, \ldots, J$:  For $i = 1, \ldots, K$:
  $\alpha_{ji} \leftarrow \frac{\sum_t Q(m_i^{(t)}=1, f^{(t)}=j)}{\sum_t Q(f^{(t)}=j)}$
  $\mu_{ji} \leftarrow \frac{\sum_t \left( Q(m_i^{(t)}=1, f^{(t)}=j) + Q(m_i^{(t)}=0) Q(b^{(t)}=j) \right) z_i^{(t)}}{\sum_t \left( Q(m_i^{(t)}=1, f^{(t)}=j) + Q(m_i^{(t)}=0) Q(b^{(t)}=j) \right)}$
  $\psi_{ki} \leftarrow \frac{\sum_t \left( Q(m_i^{(t)}=1, f^{(t)}=j) + Q(m_i^{(t)}=0) Q(b^{(t)}=j) \right) (z_i^{(t)} - \mu_{ji})^2}{\sum_t \left( Q(m_i^{(t)}=1, f^{(t)}=j) + Q(m_i^{(t)}=0) Q(b^{(t)}=j) \right)}$
```

Sum-Product EM

E Step

```
For $t = 1, \ldots, T$:
  For $i = 1, \ldots, K$: $\lambda_i^f(f) \leftarrow c_1 \left( \alpha_{fi} \mathcal{N}(z_i; \mu_{fi}, \psi_{fi}) \right.$
                            $\left. + (1 - \alpha_{fi}) \sum_b \rho_i^b(b) \mathcal{N}(z_i; \mu_{bi}, \psi_{bi}) \right)$
  $Q(f) \leftarrow c_2\ \pi_f \prod_i \lambda_i^f(f)$
  For $i = 1, \ldots, K$: $\rho_i^f(f) \leftarrow c_3\ Q(f)/\lambda_i^f(f)$
  For $i = 1, \ldots, K$:
    $\lambda_i^m(1) \leftarrow c_4\ \sum_f \rho_i^f(f) \alpha_{fi} \mathcal{N}(z_i; \mu_{fi}, \psi_{fi})$
    $\lambda_i^m(0) \leftarrow c_4\ \left( \sum_f \rho_i^f(f)(1 - \alpha_{fi}) \right) \left( \sum_b \rho_i^b(b) \mathcal{N}(z_i; \mu_{bi}, \psi_{bi}) \right)$
    $Q(m_i) \leftarrow \lambda_i^m(m_i)$
  For $i = 1, \ldots, K$: $\lambda_i^b(b) \leftarrow c_5 \left( \left( \sum_f \rho_i^f(f) \alpha_{fi} \mathcal{N}(z_i; \mu_{fi}, \psi_{fi}) \right) \right.$
                            $\left. + \left( \sum_f \rho_i^f(f)(1 - \alpha_{fi}) \right) \mathcal{N}(z_i; \mu_{bi}, \psi_{bi}) \right)$
  $Q(b) \leftarrow c_6\ \pi_b \prod_i \lambda_i^b(b)$
  For $i = 1, \ldots, K$: $\rho_i^b(b) \leftarrow Q(b)/\lambda_i^b(b)$
```

M Step

```
For $j = 1, \ldots, J$: $\pi_j \leftarrow \frac{1}{2T} (\sum_t Q(f^{(t)} = j) + \sum_t Q(b^{(t)} = j))$
For $j = 1, \ldots, J$:  For $i = 1, \ldots, K$:
  $\alpha_{ji} \leftarrow \frac{\sum_t Q(m_i^{(t)}=1) Q(f^{(t)}=j)}{\sum_t Q(f^{(t)}=j)}$
  $\mu_{ji} \leftarrow \frac{\sum_t \left( Q(m_i^{(t)}=1) Q(f^{(t)}=j) + Q(m_i^{(t)}=0) Q(b^{(t)}=j) \right) z_i^{(t)}}{\sum_t \left( Q(m_i^{(t)}=1) Q(f^{(t)}=j) + Q(m_i^{(t)}=0) Q(b^{(t)}=j) \right)}$
  $\psi_{ki} \leftarrow \frac{\sum_t \left( Q(m_i^{(t)}=1) Q(f^{(t)}=j) + Q(m_i^{(t)}=0) Q(b^{(t)}=j) \right) (z_i^{(t)} - \mu_{ji})^2}{\sum_t \left( Q(m_i^{(t)}=1) Q(f^{(t)}=j) + Q(m_i^{(t)}=0) Q(b^{(t)}=j) \right)}$
```

Fig. 5 continued.

Sec. 5.4, $P(b,f,m|z)$ is represented in the form $P(b,f|z)\prod_i P(m_i|b,f,z)$. This distribution is used in the M step to minimize the free energy w.r.t. the model parameters, $\theta = \{\alpha_k, \mu_k, \psi_k, \pi_k\}_{k=1}^K$. The resulting updates are given in Fig. 5, where we have dropped the training case index in the E step for brevity, and the constant c is computed to normalize the appropriate distribution. Starting with random parameters, the E and M steps are iterated until convergence or for a fixed number of iterations.

5.10 Generalized EM

The above derivation of the EM algorithm makes obvious several generalizations, all of which attempt to decrease $F(Q,P)$ [33]. If $F(Q,P)$ is a complex function of the parameters θ, it may not be possible to exactly solve for the θ that minimizes $F(Q,P)$ in the M step. Instead, θ can be modified so as to decrease $F(Q,P)$, *e.g.*, by taking a step downhill in the gradient of $F(Q,P)$. Or, if θ contains many parameters, it may be that $F(Q,P)$ can be optimized w.r.t. one parameter while holding the others constant. Although doing this does not solve the system of equations, it does decrease $F(Q,P)$.

Another generalization of EM arises when the posterior distribution over the hidden RVs is too complex to perform the exact update $Q(h^{(t)}) \leftarrow P(h^{(t)}|v^{(t)},\hat{\theta})$ that minimizes $F(Q,P)$ in the E step. Instead, the distribution $Q(h^{(t)})$ from the previous E step can be modified to decrease $F(Q,P)$. In fact, ICM is a special case of EM where in the E step, $F(Q,P)$ is decreased by finding the value of $\hat{h}^{(t)}$ that minimizes $F(Q,P)$ subject to $Q(h^{(t)}) = \delta(h^{(t)} - \hat{h}^{(t)})$.

5.11 Gibbs Sampling and Monte Carlo Methods

Gibbs sampling is similar to ICM, but to circumvent the local minima, *Gibbs sampling* stochastically selects the value of h_i at each step, instead of picking the MAP value of h_i:

Initialization. Pick values for all hidden RVs h (randomly, or cleverly).

Gibbs Sampling Step. Consider one of the hidden RVs, h_i. Holding all other RVs constant, sample h_i:

$$h_i \sim P(h_i|h \setminus h_i, v) = f(h_i, x_{M_i}) / \Big(\sum_{h_i} f(h_i, x_{M_i})\Big).$$

where x_{M_i} are the RVs in the Markov blanket of h_i and $f(h_i, x_{M_i})$ is the product of all conditional distributions or potentials that depend on h_i.

Repeat for a fixed number of iterations or until convergence.

Algorithmically, this is a minor modification of ICM, but in many applications it is able to escape poor local minima (c.f. [15, 19]). Also, the stochastically chosen values of h_i can be monitored to estimate the uncertainty in h_i under the posterior.

If n counts the number of sampling steps, then, as $n \to \infty$, the nth configuration of the hidden RVs is guaranteed to be an unbiased sample from the exact posterior, $P(h|v)$. In fact, although a single Gibbs sampler is not guaranteed to minimize the free energy, an infinite ensemble of Gibbs samplers *does* minimize free energy, regardless of the initial distribution of the ensemble. Let $Q^n(h)$ be the distribution over h given by the ensemble of samples at step n. Suppose we obtain a new ensemble by sampling h_i in each sampler. Then, $Q^{n+1}(h) = Q^n(h \setminus h_i)P(h_i|h \setminus h_i, v)$. Substituting Q^n and Q^{n+1} into (6), we find that $F^{n+1} \leq F^n$.

Generally, in a Monte Carlo method, the distribution over h is represented by a set of configurations $h^1, \dots, h^S$. Then, the expected value of any function of the hidden RVs, $f(h)$, is approximated by $\mathrm{E}[f(h)] \approx \frac{1}{S}\sum_{s=1}^{S} f(h^s)$. For example, if h contains binary (0/1) RVs and $h^1, \dots, h^S$ are drawn from $P(h|v)$, then by selecting $f(h) = h_i$ the above equation gives an estimate of $P(h_i = 1|v)$. There are many approaches to sampling, but the two general classes of samplers are exact samplers and Markov chain Monte Carlo (MCMC) samplers (c.f. [32]). Whereas exact samplers produce a configuration with probability equal to the probability under the model, MCMC samplers produce a sequence of configurations such that in the limit the configuration is a sample from the model. If a model $P(h, v)$ is described by a BN, then an exact sample of h *and* v can be obtained by successively sampling each RV given its parents, starting with parent-less RVs and finishing with child-less RVs. Gibbs sampling is an example of an MCMC technique.

MCMC techniques and Gibbs sampling in particular are guaranteed to produce samples from the probability model only after the memory of the initial configuration has vanished and the sampler has reached equilibrium. For this reason, the sampler is often allowed to "burn in" before samples are used to compute Monte Carlo estimates. This corresponds to discarding the samples obtained early on.

Gibbs sampling for EM in the occlusion model

Here, we describe a learning algorithm that uses ICM-updates for the model parameters, but uses stochastic updates for the RVs. This technique can be viewed as a generalized EM algorithm, where the E-Step is approximated by a Gibbs sampler. Replacing the MAP RV updates in ICM with sampling, we obtain the algorithm in Fig. 5. The notation sample_b indicates the expression on the right should be normalized w.r.t. b and then b should be sampled.

5.12 Variational Techniques and the Mean Field Method

A problem with ICM and Gibbs sampling is that when updating a particular RV, they do not account for uncertainty in the neighboring RVs. Clearly, a neighbor that is untrustworthy should count for less when updating an RV. If exact EM can be applied, then at least the exact posterior distribution is used for a subset of the RVs. However, exact EM is often not possible because the exact posterior is intractable. Also, exact EM does not account for uncertainty in the parameters.

Variational techniques assume that $Q(h)$ comes from a restricted family of distributions that can be efficiently searched over. Inference proceeds by minimizing $F(Q, P)$ w.r.t. $Q(h)$, but the restriction on $Q(h)$ implies that a tight bound, $F = -\ln P(v)$, is not in general achievable. In practice, the family of distributions is usually chosen so that a closed form expression for $F(Q, P)$ can be obtained and optimized.

The "starting point" when deriving variational techniques is the product form (a.k.a. fully-factorized, or mean-field) Q-distribution. If h consists of M hidden RVs $h = (h_1, \ldots, h_M)$, the product form Q distribution is

$$Q(h) = \prod_{i=1}^{M} Q(h_i), \tag{9}$$

where there is one variational parameter or one set of variational parameters that specifies the marginal $Q(h_i)$ for each hidden RV h_i.

The advantage of the product form approximation is most readily seen when $P(h, v)$ is described by a BN. Suppose the kth conditional probability function is a function of RVs h_{C_k} and v_{D_k} and denote it by $g_k(h_{C_k}, v_{D_k})$. So, $P(h, v) = \prod_k g_k(h_{C_k}, v_{D_k})$. Substituting this and (9) into (6), we obtain the mean field free energy:

$$F(Q, P) = \sum_i \Big(\int_{h_i} Q(h_i) \ln Q(h_i) \Big) - \sum_k \Big(\int_{h_{C_k}} \Big(\prod_{i \in C_k} Q(h_i) \Big) \ln g_k(h_{C_k}, v_{D_k}) \Big).$$

The high-dimensional integral over all hidden RVs simplifies into a sum over the conditional probability functions, of low-dimensional integrals over small collections of hidden RVs. The first term is the sum of the negative entropies of the Q-distributions for individual hidden RVs. For many scalar RVs (*e.g.*, Bernoulli, Gaussian, *etc.*) the entropy can be written in closed form quite easily.

The second term is the sum of the expected log-conditional distributions, where for each conditional distribution, the expectation is taken with respect to the product of the Q-distributions for the hidden RVs. For appropriate forms of the conditional distributions, this term can also be written in closed form. For example, suppose $P(h_1|h_2) = exp(-\ln(2\pi\sigma^2)/2 - (h_1 - ah_2)^2/2\sigma^2)$ (*i.e.*, h_1 is Gaussian with mean

ah_2), and $Q(h_1)$ and $Q(h_2)$ are Gaussian with means ϕ_{11} and ϕ_{21} and variances ϕ_{12} and ϕ_{22}. The entropy terms for h_1 and h_2 are $-\ln(2\pi e\phi_{12})/2$ and $-\ln(2\pi e\phi_{22})/2$. The other term is the expected value of a quadratic form under a Gaussian, which is straightforward to compute. The result is $-\ln(2\pi\sigma^2)/2-(\phi_{11}-a\phi_{21})^2/2\sigma^2-\phi_{12}/2\sigma^2-a^2\phi_{22}/2\sigma^2$. These expressions are easily-computed functions of the variational parameters. Their derivatives (needed for minimizing $F(Q,P)$) can also be computed quite easily.

In general, variational inference consists of searching for the value of ϕ that minimizes $F(Q,P)$. For convex problems, this optimization is easy. Usually, $F(Q,P)$ is not convex in Q and iterative optimization is required:

Initialization. Pick values for the variational parameters, ϕ (randomly, or cleverly).

Optimization Step. Decrease $F(Q,P)$ by adjusting the parameter vector ϕ, or a subset of ϕ.

Repeat for a fixed number of iterations or until convergence.

This variational technique accounts for uncertainty in *both* the hidden RVs and the hidden model parameters. If the amount of training data is small, a variational approximation to the parameters can be used to represent uncertainty in the model due to the sparse training data.

Often, variational techniques are used to approximate the distribution over the hidden RVs in the E step of the EM algorithm, but point estimates are used for the model parameters. In such *variational EM algorithms*, the Q-distribution is $Q(h)=\delta(\theta-\hat{\theta})\prod_{t=1}^{T}Q(h^{(t)};\phi^{(t)})$. Note that there is one set of variational parameters for each training case. In this case, we have the following generalized EM steps:

Initialization. Pick values for the variational parameters $\phi^{(1)},\ldots,\phi^{(T)}$ and the model parameters $\hat{\theta}$ (randomly, or cleverly).

Generalized E Step. Starting from the variational parameters from the previous iteration, modify $\phi^{(1)},\ldots,\phi^{(T)}$ so as to decrease F.

Generalized M Step. Starting from the model parameters from the previous iteration, modify $\hat{\theta}$ so as to decrease F.

Repeat for a fixed number of iterations or until convergence.

Variational inference for EM in the occlusion model

The fully-factorized Q-distribution over the hidden RVs for a single data sample in the occlusion model is $Q(m,f,b)=Q(b)Q(f)\prod_{i=1}^{K}Q(m_i)$. Substituting this Q-distribution into the free energy for a single observed data sample in the occlusion model, we obtain

$$
\begin{aligned}
F =& \sum_b Q(b) \ln \frac{Q(b)}{\pi_b} + \sum_f Q(f) \ln \frac{Q(f)}{\pi_f} \\
&+ \sum_i \sum_f Q(f) \Big(Q(m_i = 1) \ln \frac{Q(m_i = 1)}{\alpha_{fi}} + Q(m_i = 0) \ln \frac{Q(m_i = 0)}{1 - \alpha_{fi}} \Big) \\
&+ \sum_i \sum_f Q(f) Q(m_i = 1) \Big(\frac{(z_i - \mu_{fi})^2}{2\psi_{fi}} + \frac{\ln 2\pi\psi_{fi}}{2} \Big) \\
&+ \sum_i Q(m_i = 0) \Big(\sum_b Q(b) \Big(\frac{(z_i - \mu_{bi})^2}{2\psi_{bi}} + \frac{\ln 2\pi\psi_{bi}}{2} \Big) \Big).
\end{aligned}
$$

The first two terms keep $Q(b)$ and $Q(f)$ close to their priors π_b and π_f. The third term keeps $Q(m_i)$ close to the mask priors α_{fi} for foreground classes that have high posterior probability $Q(f)$. The last two terms favor mask values and foreground/background classes that minimize the variance-normalized squared differences between the predicted pixel values and the observed pixel values.

Setting the derivatives of F to zero, we obtain the updates for the Q-distributions in the E step. Once the variational parameters are computed for all observed images, the total free energy $F = \sum_t F^{(t)}$ is optimized w.r.t. the model parameters to obtain the variational M step. The resulting updates are given in Fig. 5. Each E step update can be computed in $O(KJ)$ time, which is a K-fold speed-up over exact inference used for exact EM. This speed-up is obtained because the variational method assumes that f and b are independent in the posterior. Also, note that if the Q-distributions place all mass on one configuration, the E step updates reduce to the ICM updates The M step updates are similar to the updates for exact EM, except that the exact posterior distributions are replaced by their factorized surrogates.

The above updates can be iterated in a variety of ways. For example, each iteration may consist of repeatedly updating the variational distributions until convergence and then updating the parameters. Or, each iteration may consist of updating each variational distribution once, and then updating the parameters. There are many possibilities and the update order that is best at avoiding local minima depends on the problem.

5.13 Structured Variational Techniques

The product-form (mean-field) approximation does not account for dependencies between hidden RVs. For example, if the posterior has two distinct modes, the variational technique for the product-form approximation will find only one mode. With a different initialization, the technique may find another mode, but the exact form of the dependence is not revealed. In structured variational techniques, the Q-

distribution is itself specified by a graphical model, such that $F(Q, P)$ can still be optimized. Fig. 6a shows the original BN for the occlusion model and Fig. 6b shows the BN for the fully-factorized (mean field) Q-distribution described above. Recall that the exact posterior can be written $P(m, f, b|z) = Q(m, f, b) = Q(f, b) \prod_{i=1}^{K} Q(m_i|f, b)$. Fig. 6c shows the BN for this Q-distribution. Generally, increasing the number of dependencies in the Q-distribution leads to more exact inference algorithms, but also increases the computational demands of variational inference. In the occlusion model, whereas mean field inference takes KJ time, exact inference takes KJ^2 time. However, additional dependencies can sometimes be accounted for at no extra computational cost. As described below, it turns out that the Q-distribution shown in Fig. 6d leads to an inference algorithm with the same complexity as the mean field method (KJ time), but can account for dependencies of the mask RVs on the foreground class.

Structured variational inference for EM in the occlusion model

The Q-distribution corresponding to the BN in Fig. 6c is $Q(m, f, b) = Q(b)Q(f) \prod_{i=1}^{K} Q(m_i|f)$. Defining $q_{fi} = Q(m_i = 1|f)$, we have $Q(m, f, b) = Q(b)Q(f) \prod_{i=1}^{K} q_{fi}^{m_i}(1 - q_{fi})^{1-m_i}$. Substituting this Q-distribution into the free energy for the occlusion model, we obtain

$$
\begin{aligned}
F = &\sum_b Q(b) \ln \frac{Q(b)}{\pi_b} + \sum_f Q(f) \ln \frac{Q(f)}{\pi_f} \\
&+ \sum_i \sum_f Q(f) \Big(Q(m_i = 1|f) \ln \frac{Q(m_i = 1|f)}{\alpha_{fi}} + Q(m_i = 0|f) \ln \frac{Q(m_i = 0|f)}{1 - \alpha_{fi}} \Big) \\
&+ \sum_i \sum_f Q(f)Q(m_i = 1|f) \Big(\frac{(z_i - \mu_{fi})^2}{2\psi_{fi}} + \frac{\ln 2\pi\psi_{fi}}{2} \Big) \\
&+ \sum_i \Big(\Big(\sum_f Q(f)Q(m_i = 0|f) \Big) \sum_b Q(b) \Big(\frac{(z_i - \mu_{bi})^2}{2\psi_{bi}} + \frac{\ln 2\pi\psi_{bi}}{2} \Big) \Big).
\end{aligned}
$$

Setting the derivatives of F to zero, we obtain the updates given in Fig. 5. With some care, these updates can be computed in $O(KJ)$ time, which is a K-fold speed-up over exact inference. Although the dependencies of f and m_i, $i = 1, \ldots, K$ on b are not accounted for, the dependence of m_i on f is accounted for by the q_{fi}'s. The parameter updates in the M step have a similar form as for exact EM, except that the exact posterior is replaced by the above, structured Q-distribution.

5.14 The Sum-Product Algorithm (Belief Propagation)

The sum-product algorithm (a.k.a. belief propagation, probability propagation) performs inference by passing messages along the edges of the graphical model (see [25] for an extensive review). The message arriving at an RV is a probability distribution (or a function that is proportional to a probability distribu-

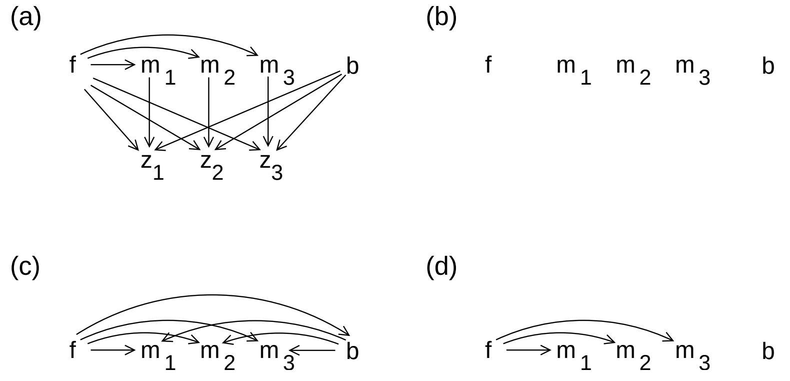


Figure 6: Starting with the BN of the original occlusion model (a), variational techniques ranging from the fully factorized approximation to exact inference can be derived. (b) The BN for the factorized (mean field) Q-distribution. z is observed, so it is not included in the graphical model for the Q-distribution. (c) The BN for a Q-distribution that can represent the *exact* posterior. (d) The BN for a Q-distribution that can represent the dependence of the mask RVs on the foreground class. Accounting for more dependencies improves the bound on the data likelihood, but the choice of which dependencies are retained has a large impact on the improvement in the bound.

tion), that represents the inference for the RV, as given by the part of the graph that the message came from. Pearl [35] showed that the algorithm is exact if the graph is a tree. When the graph contains cycles, the sum-product algorithm (a.k.a. "loopy belief propagation") is not exact and can diverge and oscillate. However, it has been used in vision algorithms [8]. Surprisingly, we have also found that its oscillatory behavior can be used to jump between modes of the posterior Also, it has produced state-of-the-art results on several difficult problems, including error-correcting decoding [14], medical diagnosis [30], *random* satisfiability [28], and phase-unwrapping in 2-dimensions [13].

To see how the sum-product algorithm works, consider computing $P(a)$ in the model $P(a,b,c,d) = P(a|b)P(b|c)P(c|d)P(d)$. One approach is to compute $P(a,b,c,d)$ for all values of a, b, c and d and then compute $P(a) = \sum_b \sum_c \sum_d P(a,b,c,d)$. For binary RVs, this takes $(3+1)(2 \cdot 2 \cdot 2 \cdot 2)$ operations. Alternatively, we can move the sums inside the products: $P(a) = \sum_b P(a|b)\{\sum_c P(b|c)[\sum_d P(c|d)P(d)]\}$. If the terms are computed from the inner-most term out, this takes $(3)(2+2+2)$ operations, giving an exponential speed-up in the number of RVs. The computation of each term in braces corresponds to the computation of a message in the sum-product algorithm.

In a graphical model, the joint distribution can be written $P(h,v) = \prod_k g_k(h_{C_k}, v_{D_k})$, where h_{C_k} and v_{D_k} are the hidden and visible RVs in the kth local function (or conditional distribution). The sum-product algorithm approximates $P(h|v)$ by $Q(h)$, where $Q(h)$ is specified by marginals $Q(h_i)$ *and* clique marginals $Q(h_{C_k})$. These are computed by combining messages that are computed iteratively in the FG. Denote the message sent from variable h_i to function g_k by $\mu_{ik}(h_i)$ and denote the message sent from function g_k to variable h_i by $\mu_{ki}(h_i)$. Note that the message passed on an edge is a function of the neighboring variable. A user-specified *message-passing schedule* is used to determine which messages should be updated at each iteration. The sum-product algorithm proceeds as follows:

Initialization. Set all messages to be uniform.

Message Update Step. Update the messages specified in the message-passing schedule. The message sent from variable h_j to function g_k is updated as follows:

$$\mu_{jk}(h_j) \leftarrow c \prod_{n: j \in C_n, n \neq k} \mu_{nj}(h_j), \tag{10}$$

where c is computed so as to normalize the message. The message sent from function g_k to variable h_j is updated as follows:

$$\mu_{kj}(h_j) \leftarrow c \sum_{h_{C_k \setminus j}} \Big(g_k(h_{C_k}, v_{D_k}) \prod_{i \in C_k, i \neq j} \mu_{ik}(h_i) \Big), \tag{11}$$

where $C_k \setminus j$ is the set of indices C_k with j removed.

Fusion. A single-variable marginal or clique marginal can be computed at any time as follows:

$$Q(h_j) \leftarrow c \prod_{n: j \in C_n} \mu_{nj}(h_j), \tag{12}$$

$$Q(h_{C_k}) \leftarrow c \;\; g_k(h_{C_k}, v_{D_k}) \prod_{i \in C_k} \mu_{ik}(h_i), \tag{13}$$

Repeat for a fixed number of iterations or until convergence.

If the graph is a tree, once messages have flowed from every node to every other node, the estimates of the posterior marginals are *exact*. So, if the graph has E edges, exact inference is accomplished by propagating $2E$ messages according to the following message-passing schedule. Select one node as the root and arrange the nodes in layers beneath the root. Propagate messages from the leaves to the root (E messages) and then propagate messages from the root to the leaves (another E messages). This procedure ensures that messages flow from every node to every other node. Note that if the graph is a tree, if normalizations are *not* performed during message-passing, the fusion equations compute the *joint* probability of the hidden variable(s) and the observed variables: $\prod_{n: j \in C_n} \mu_{nj}(h_j) = P(h_j, v)$.

If the graph contains cycles, messages can be passed in an iterative fashion for a fixed number of iterations, until convergence is detected, or until divergence is detected. Also, various schedules for updating the messages can be used and the quality of the results will depend on the schedule. It is proven in [37] that when the "max-product" algorithm converges, all configurations that differ by perturbing the RVs in subgraphs that contain at most one cycle, will have *lower* posterior probabilities.

If the graphical model is a BN, so that $\mathcal{Z} = 1$, the sum-product algorithm can be used for inference in a generalized EM algorithm as follows:

Initialization. Pick values for the model parameters θ (randomly, or cleverly), and set all messages to be uniform.

Generalized E Step. For each training case $v^{(t)}$, apply one or more iterations of the sum-product algorithm. Then, fuse messages as described above to compute $Q(h^{(t)}_{C_k})$ for every child and its parents.

Generalized M Step. Modify the parameters θ so as to maximize

$$\sum_t \sum_k \sum_{h^{(t)}_{C_k}} Q(h^{(t)}_{C_k}) \ln g_k(h^{(t)}_{C_k}, v^{(t)}_{D_k}; \theta).$$

Repeat for a fixed number of iterations or until convergence.

The sum-product algorithm for EM in the occlusion model

For an occlusion model with K pixels, exact inference takes $O(KJ^2)$ time. In contrast, loopy belief propagation takes $O(KJ)$ time, assuming the number of iterations needed for convergence is constant. Generally, the computational gain from using loopy belief propagation is exponential in the number of RVs that combine to explain the data.

The graphical model has cycles, so before applying the sum-product algorithm, we modify it to reduce the number of cycles, as shown in in Fig. 7a, where the observed pixels $z_1, \ldots, z_K$ are not shown for visual clarity. For each pixel i, there is *one* local function g_i that combines the conditional distributions for each mask RV and its corresponding pixel:

$$g_i(f, b, m_i) = P(z_i|m_i, f, b)P(m_i|f) = \mathcal{N}(z_i; \mu_{fi}, \psi_{fi})^{m_i}\mathcal{N}(z_i; \mu_{bi}, \psi_{bi})^{1-m_i}\alpha_{fi}^{m_i}(1-\alpha_{fi})^{1-m_i}.$$

Fig. 7b shows how we have labeled the messages along the edges of the FG. During message passing, some messages will always be the same. In particular, a message leaving a singly-connected function node will always be equal to the function. So, the messages leaving the nodes corresponding to $P(f)$ and $P(b)$ are equal to $P(f)$ and $P(b)$, as shown in Fig. 7b. Also, a message leaving a singly-connected variable node will always be equal to the constant 1. So, the messages leaving the mask RVs, m_i are 1. Initially, all other messages are set to the value 1.

Before updating messages in the graph, we must specify in what order the messages should be updated.

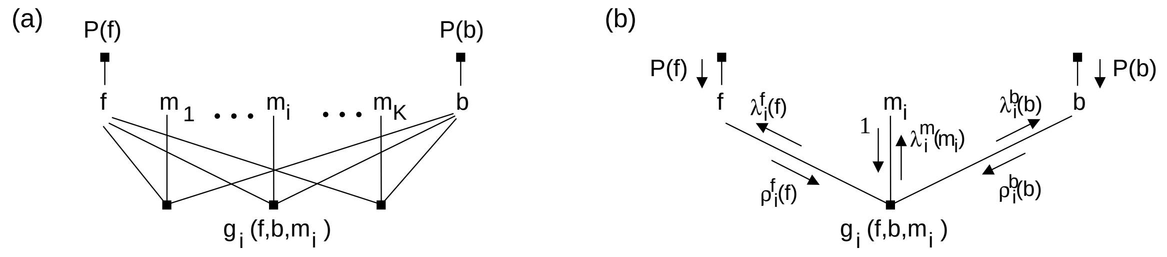


Figure 7: (a) The FG for the occlusion model with K pixels, after the observations $(z_1, \ldots, z_K)$ are absorbed into function nodes, $g_i(f, b, m_i) = P(z_i|m_i, f, b)P(m_i|f)$. (b) The sum-product algorithm (belief propagation) passes messages along each edge of the graph. This graph fragment shows the different types of messages propagated in the occlusion model.

This choice will influence how quickly the algorithm converges, and for graphs with cycles can influence whether or not it converges at all. Messages can be passed until convergence, or for a fixed amount of time. Here, we define one iteration to consist of passing messages from the g's to b, from b to the g's, from the g's to f, from f to the g's, and from the g's to the m's. Each iteration ensures that each RV propagates its influence to every other RV. Since the graph has cycles, this procedure should be repeated.

The message updates are derived from the general rules described above. From (11), it is straightforward to show that the message sent from g_i to f should be updated as follows: $\lambda_i^f(f) \leftarrow \sum_b \sum_{m_i} \rho_i^b(b) \cdot 1 \cdot g_i(f, b, m_i)$. Note that since the resulting message is a function of f alone, b and m_i must be summed over. Substituting $g_i(f, b, m_i)$ from above and assuming that $\rho_i^b(b)$ is normalized, this update can be simplified: $\lambda_i^f(f) \leftarrow \alpha_{fi}\mathcal{N}(z_i; \mu_{fi}, \psi_{fi}) + (1 - \alpha_{fi}) \sum_b \rho_i^b(b)\mathcal{N}(z_i; \mu_{bi}, \psi_{bi})$. The last step in computing this message is to normalize it: $\lambda_i^f(f) \leftarrow \lambda_i^f(f)/(\sum_f \lambda_i^f(f))$.

According to (10), the message sent from f to g_i is given by the product of the other incoming messages, $\rho_i^f(f) \leftarrow \pi_f \prod_{j \neq i} \lambda_i^f(f)$, and it is then normalized: $\rho_i^f(f) \leftarrow \rho_i^f(f)/(\sum_f \rho_i^f(f)$.

The message sent from g_i to b is given by $\lambda_i^b(b) \leftarrow \sum_f \sum_{m_i} \rho_i^f(f) \cdot 1 \cdot g_i(f, b, m_i)$, which simplifies to $\lambda_i^b(b) \leftarrow \big(\sum_f \rho_i^f(f)\alpha_{fi}\mathcal{N}(z_i; \mu_{fi}, \psi_{fi})\big) + \big(\sum_f \rho_i^f(f)(1 - \alpha_{fi})\big)\mathcal{N}(z_i; \mu_{bi}, \psi_{bi})$. Note that the terms in large parentheses don't depend on b, so they need to be computed only once when updating this message. Again, before proceeding, the message is normalized: $\lambda_i^b(b) \leftarrow \lambda_i^b(b)/(\sum_b \lambda_i^b(b))$.

The message sent from b to g_i is given by $\rho_i^b(b) \leftarrow \pi_b \prod_{j \neq i} \lambda_i^b(b)$, and then normalized: $\rho_i^b(b) \leftarrow \rho_i^b(b)/(\sum_b \rho_i^b(b)$.

Finally, the message sent from g_i to m_i is updated as follows: $\lambda_i^m(m_i) \leftarrow \sum_f \sum_b \rho_i^f(f) \cdot \rho_i^b(b) \cdot g_i(f, b, m_i)$. For $m_i = 1$ and $m_i = 0$ this update simplifies to $\lambda_i^m(1) \leftarrow \sum_f \rho_i^f(f)\alpha_{fi}\mathcal{N}(z_i; \mu_{fi}, \psi_{fi})$ and $\lambda_i^m(0) \leftarrow \big(\sum_f \rho_i^f(f)(1 - \alpha_{fi})\big)\big(\sum_b \rho_i^b(b)\mathcal{N}(z_i; \mu_{bi}, \psi_{bi})\big)$. Normalization is performed by setting $\lambda_i^m(m_i) \leftarrow \lambda_i^m(m_i)/(\lambda_i^m(0) + \lambda_i^m(1))$.

At any point during message-passing, the fusion rule in (12) can be used to estimate posterior marginals of variables. The estimates of $P(f|z)$, $P(b|z)$ and $P(m_i|z)$ are $Q(f) \leftarrow \big(\pi_f \prod_i \lambda_i^f(f)\big)/\big(\sum_f \pi_f \prod_i \lambda_i^f(f)\big)$, $Q(b) \leftarrow \big(\pi_b \prod_i \lambda_i^b(b)\big)/\big(\sum_b \pi_b \prod_i \lambda_i^b(b)\big)$, and $Q(m_i) \leftarrow \lambda_i^m(m_i)$. It is common to compute these during each iteration. In fact, computing the posterior marginals is often useful as an intermediate step for more efficiently computing other messages. For example, direct implementation of the above updates for $\rho_i^f(f)$ requires order JK^2 time. However, if $Q(f)$ is computed first (which takes order JK time), then $\rho_i^f(f)$ can be updated in order JK time using $\rho_i^f(f) \leftarrow Q(f)/\lambda_i^f(f)$, followed by normalization.

Fig. 5 shows the generalized EM algorithm where the E step uses the sum-product algorithm. Whereas algorithms presented earlier have one update for each variable (whether in terms of its value or its distribution), the sum-product algorithm has one update for each edge in the graph. Note that when updating $Q(b)$ and $Q(f)$, whereas variational methods adjust the effect of each likelihood term by raising it to a power, the sum-product algorithm adds an offset that depends on how well the other hidden variables account for the data. In the M step, we have used a factorized approximation to $Q(m_i, f)$ and $Q(m_i, b)$. In fact, these clique marginals can be computed using (13), to obtain a more exact M step.

The sum-product algorithm as a variational method

The sum-product algorithm can be thought of as a variational technique. Recall that in contrast to product-form variational techniques, structured variational techniques account for more of the direct dependencies (edges) in the original graphical model, by finding Q-distributions over disjoint substructures (sub-graphs). However, one problem with structured variational techniques is that dependencies induced by the edges that connect the sub-graphs are accounted for quite weakly through the variational parameters in the Q-distributions for the sub-graphs. In contrast, the sum-product algorithm uses a set of sub-graphs that cover *all* edges in the original graph and accounts for every direct dependence *approximately*, using one or more Q-distributions.

To derive the sum-product algorithm as a variational method we follow [38]. As described earlier, the sum-product algorithm approximates $P(h|v)$ by $Q(h)$, where $Q(h)$ is specified by marginals $Q(h_i)$ and clique marginals $Q(h_{C_k})$. Notice that these sets of marginals cover *all* edges in the graphical model. Substituting the expression for $P(h, v)$ into (6) the free energy is $F = \sum_h Q(h) \ln Q(h) - \sum_k \sum_{h_{C_k}} Q(h_{C_k}) \ln g_k(h_{C_k}, v_{D_k})$. The second term is a local expectation that can usually be computed or approximated efficiently. However, since we don't have a factorized expression for $Q(h)$, the first term is generally intractable. We can approximate $Q(h)$ inside the logarithm using the *Bethe approximation*: $Q(h) \approx (\prod_k Q(h_{C_k}))/(\prod_i Q(h_i)^{d_i-1})$, where d_i is the degree of h_i, *i.e.*, the number of terms $Q(h_{C_k})$ that h_i appears in. The denominator is meant to account for the overlap between the clique marginals. For trees, the Bethe approximation is exact (c.f. [26]).

Substituting the Bethe approximation for the term $\ln Q(h)$, we obtain the Bethe free energy F_{Bethe}, which *approximates* the true free energy, $F_{\mathrm{Bethe}} \approx F$:

$$F_{\mathrm{Bethe}} = \sum_k \sum_{h_{C_k}} Q(h_{C_k}) \ln Q(h_{C_k}) - \sum_i (d_i - 1) \sum_{h_i} Q(h_i) \ln Q(h_i) - \sum_k \sum_{h_{C_k}} Q(h_{C_k}) \ln g_k(h_{C_k}, v_{D_k}).$$

This approximation becomes exact if the graph is a tree. If the graph is not a tree, we can still try to minimize F_{Bethe} w.r.t. $Q(h_{C_k})$ and $Q(h_i)$, but during optimization the marginals are usually not consistent with any probability distribution on h. The statistical physics community has developed more complex, but more accurate approximations, such as the Kikuchi approximation, which can be used to derive inference algorithms [38].

The minimization of F_{Bethe} must account for the marginalization constraints, $\forall k : \sum_{h_{C_k}} Q(h_{C_k}) = 1$, $\forall i : \sum_{h_i} Q(h_i) = 1$, and $\forall k, \forall i \in C_k : \sum_{h_{C_k \setminus i}} Q(h_{C_k}) = Q(h_i)$, where $C_k \setminus i$ is the set of indices C_k with i removed. The last constraint ensures that the single-variable marginals and the clique marginals agree. Denote the Lagrange multipliers for these constraints by υ_k, ν_i and $\gamma_{ik}(h_i)$, where the last multiplier depends on the value h_i, since there is one constraint for each value of h_i. Setting the derivatives of F_{Bethe} subject to these constraints to 0, we obtain $Q(h_j)^{d_j - 1} \propto \prod_{k:j \in C_k} e^{\gamma_{jk}(h_j)}$ and $Q(h_{C_k}) \propto g_k(h_{C_k}, v_{D_k}) \prod_{i \in C_k} e^{\gamma_{ik}(h_i)}$.

The sum-product algorithm can be viewed as an algorithm that recursively computes the Lagrange multipliers, $\gamma_{ik}(h_i)$, so as to satisfy the above two equations and the marginalization constraint everywhere in the network. In the standard form of the sum-product algorithm, we define $\mu_{ik}(h_i) = e^{\gamma_{ik}(h_i)}$ to be a "message" sent from variable h_i to function g_k. Using this notation, the equations and the marginalization constraint give the following system of equations: $Q(h_j)^{d_j - 1} \propto \prod_{k:i \in C_k} \mu_{jk}(h_j)$, $Q(h_{C_k}) \propto g_k(h_{C_k}, v_{D_k}) \prod_{i \in C_k} \mu_{ik}(h_i)$, and $\sum_{h_{C_k \setminus i}} Q(h_{C_k}) = Q(h_i)$.

One way of solving the system is to find a set of update equations whose fixed points satisfy the system. To do this, introduce "messages" that are sent from functions to variables: $\mu_{kj}(h_j)$ is a message sent from function g_k to variable h_j. A fixed point of the sum-product updates in (10) to (13) satisfies the system of equations. From (10), we have $\prod_{k:i \in C_k} \mu_{jk}(h_j) = \prod_{k:i \in C_k} \prod_{n:j \in C_n, n \neq k} \mu_{nj}(h_j) = \left(\prod_{n:j \in C_n} \mu_{nj}(h_j) \right)^{d_j - 1}$. Combining this with (12) we obtain $\prod_{k:i \in C_k} \mu_{jk}(h_j) = Q(h_j)^{d_j - 1}$ which satisfies the first equation in the system. The second equation is satisfied trivially by sum-product update (13). To see how the third equation is satisfied, first sum over $h_{C_k \setminus j}$ in (13) and then use (11) to obtain $\sum_{h_{C_k \setminus j}} Q(h_{C_k}) \propto \mu_{jk}(h_j) \mu_{kj}(h_j)$. Then, substitute $\mu_{jk}(h_j)$ from (10) and use (12) to obtain $\sum_{h_{C_k \setminus j}} Q(h_{C_k}) \propto \prod_{n:j \in C_n} \mu_{nj}(h_j) \propto Q(h_j)$, which satisfies the third equation.

Method	Update for mask variables	Complexity
Exact inference (used in EM)	$\frac{Q(m_i=1\|b,f)}{Q(m_i=0\|b,f)} \leftarrow \frac{\alpha_{fi}\mathcal{N}(z_i;\mu_{fi},\psi_{fi})}{(1-\alpha_{fi})\mathcal{N}(z_i;\mu_{bi},\psi_{bi})}$	J^2K
Iterative conditional modes	$m_i \leftarrow \begin{cases} 1, & \text{if } \frac{\alpha_{fi}\mathcal{N}(z_i;\mu_{fi}\psi_{fi})}{(1-\alpha_{fi})\mathcal{N}(z_i;\mu_{bi},\psi_{bi})} > 1 \\ 0, & \text{otherwise} \end{cases}$	K
Gibbs sampling	$m_i \leftarrow \text{sample}_{m_i} \left\{ \begin{array}{ll} \alpha_{fi}\mathcal{N}(z_i;\mu_{fi},\psi_{fi}) & \text{if } m_i = 1 \\ (1-\alpha_{fi})\mathcal{N}(z_i;\mu_{bi},\psi_{bi}) & \text{if } m_i = 0 \end{array} \right\}$	K
Mean field	$\frac{Q(m_i=1)}{Q(m_i=0)} \leftarrow \frac{\prod_f(\alpha_{fi}\mathcal{N}(z_i;\mu_{fi},\psi_{fi}))^{Q(f)}}{(\prod_f(1-\alpha_{fi})^{Q(f)})(\prod_b \mathcal{N}(z_i;\mu_{bi},\psi_{bi})^{Q(b)})}$	JK
Structured variational	$\frac{Q(m_i=1\|f)}{Q(m_i=0\|f)} \leftarrow \frac{\alpha_{fi}\mathcal{N}(z_i;\mu_{fi},\psi_{fi})}{(1-\alpha_{fi})\prod_b \mathcal{N}(z_i;\mu_{bi},\psi_{bi})^{Q(b)}}$	JK
Sum-product algorithm	$\frac{Q(m_i=1)}{Q(m_i=0)} \leftarrow \frac{\sum_f \rho_i^f(f)\alpha_{fi}\mathcal{N}(z_i;\mu_{fi},\psi_{fi})}{(\sum_f \rho_i^f(f)(1-\alpha_{fi}))(\sum_b \rho_i^b(b)\mathcal{N}(z_i;\mu_{bi},\psi_{bi}))}$	JK

Table 1: A comparison of the updates for the mask variables for various algorithms discussed in this tutorial.

5.15 Annealing

In all of the above techniques, when searching for $Q(h)$, local minima of F can be a problem. One way to try to avoid local minima is to introduce an inverse temperature, β: $F(\beta) = \int_h Q(h) \ln Q(h)/P(h,v)^\beta$. When $\beta = 0$, $P(h,v)^\beta$ is uniform and inference is easy. When $\beta = 1$, $P(h,v)^\beta = P(h,v)$ and $F(\beta) = F$, the free energy we want to minimize. By searching over Q while annealing the system – adjusting β from 0 to 1 – the search *may* avoid local minima. In practice, the use of annealing raises the difficult question of how to adjust β during inference.

6 Comparison of Algorithms

Each of the above techniques iteratively updates an approximation to the exact posterior distribution while searching for a minimum of the free energy. It is useful to study how the behaviors of the algorithms differ. In Table 1, we give the update equations for the mask variables in the occlusion model. These updates have been written in a slightly different form than presented in Fig. 5, to make comparisons between different methods easier.

Whereas exact inference computes the distribution over m_i for every configuration of the neighboring variables b and f, ICM and Gibbs sampling select a new value of m_i based on the *single* current configuration of b and f. Whereas updating all mask variables takes J^2K time for exact inference, it takes K time for ICM and Gibbs sampling.

The update for the distribution $Q(m_i)$ over m_i in the mean field (fully-factorized) variational method can be compared to the update for exact inference. The updates are similar, but an important difference is that each term that depends on f or b is replaced by its geometric average w.r.t. the current distribution $Q(f)$ or $Q(b)$. Each such geometric average takes J time and there are K mask variables, so updating all mask variables takes JK time.

In the structured variational method, the dependence of m_i on f is taken into account. The update for the distribution $Q(m_i|f)$ is similar to the update in the mean field method, but the geometric averages for terms that depend on f are not taken (since one Q-distribution is computed for each value of f). The term that depends on b does not depend on f, so its geometric average w.r.t. b can be computed once for all f. The resulting updates for all mask variables takes JK time, which is the same as for the mean field method. This example shows that sometimes, accounting for more dependencies does not significantly increase the time-complexity of a variational method.

Comparing the update for $Q(m_i)$ in the sum-product algorithm with the corresponding update in the fully-factorized variational method, we see that the geometric averages are replaced with arithmetic averages. This is an important difference between the two methods. While the geometric average favors values of m_i that have high weight in all terms, the arithmetic average favors values of m_i that have high weight in at least 1 term. In this sense, the sum-product algorithm is more "inclusive" of possible configurations of hidden variables, than fully-factorized variational methods. Another difference between these two methods is that while the variational method takes averages w.r.t. the same distribution for all pixels, $Q(f)$ or $Q(b)$, the sum-product algorithm uses pixel-specific distributions, $\rho_i^f(f)$ or $\rho_i^b(b)$.

7 Experimental Results

We explored the following algorithms for learning the parameters of the occlusion model described in Sec. 2.1, using the data illustrated in Fig. 1: ICM, exact EM, Gibbs sampling; variational EM with a fully-factorized posterior, structured variational EM, and the sum-product algorithm for EM. The MATLAB scripts we used are available on our web sites.

We found that the structured variational method performed almost identically to the fully-factorized variational method, so we do not report results on the structured variational method. Generally, there usually are structured variational approximations that produce bounds that are significantly better than mean field, but are much more computationally efficient than exact inference (c.f. [12]).

Each technique can be tweaked in a variety of ways to improve performance. However, our goal is to provide the reader with a "peek under the hood" of each inference engine and convey a qualitative sense of the similarities and differences between the techniques, so we strove to make the initial conditions, variable/parameter update schedules, *etc.* as similar as possible. For details of training conditions, see the MATLAB scripts posted on our web sites.

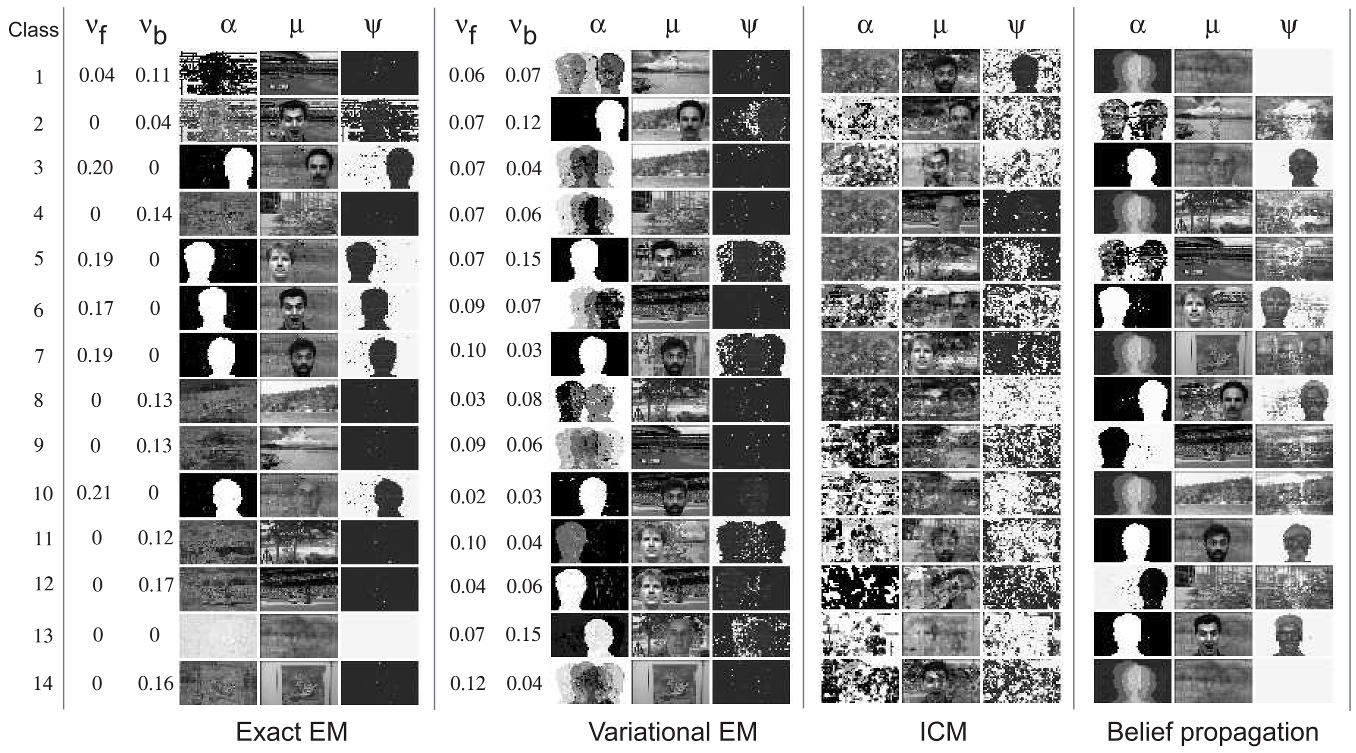
The learning algorithms are at best guaranteed to converge to a *local* minimum of the free energy, which is an upper bound on the negative log-likelihood of the data. A common local minimum is a set of images in which some of the true classes in the data are repeated while the others are merged into blurry images. To help avoid this type of local minimum, we provided the model with 14 clusters – 2 more than the total number of different foregrounds and backgrounds. (If too many clusters are used, the model tends to overfit and learn specific combinations of foreground and background.)

Each learning algorithm was run 5 times with different random initializations and the run with the highest log-likelihood was kept. For complex models, computing the log-likelihood is intractable and the free energy should be used instead. The pixels in the class means were initialized to independent values drawn from the uniform density in $[0, 1)$, the pixel variances were set to 1, and the mask probability for each pixel was set to 0.5. All classes were allowed to be used in both foreground and background images. To avoid numerical problems, the model variances and the prior and posterior probabilities on discrete RVs f, b, m_i were not allowed to drop below 10^{-6}.

Fig. 8 shows the parameters after convergence of the learning algorithms, and Fig. 9 shows the free energy as a function of the number of computations needed during learning. Most techniques managed to find all classes of appearance, but the computational requirements varied by 2 orders of magnitude. However, the greediest technique, ICM, failed to find all classes. The ability to disambiguate foreground and background classes is indicated by the estimated mask probabilities α (see also the example in Fig. 11), as well as the total posterior probability of a class being used as a background (ν^b), and foreground (ν^f).

Exact EM for the most part correctly infers which of the classes are used as foreground or background. The only error it made is evident in the first two learned classes, which are sometimes swapped to model the combination of the background and foreground layers, shown in the last example from the training set in Fig. 1. This particular combination (12 images in the dataset) is modeled with class 2 in the background and class 1 in the foreground. This is a consequence of using 14 classes, rather than the required 12 classes. Without class 2, which is a repeated version of class 6, class 6 would be correctly used as a foreground class for these examples. The other redundancy is class 13, which ends up with a probability close to zero, indicating it is not used by the model.

The variational technique does not properly disambiguate foreground from background classes, as is evident from the total posterior probabilities of using a class in each layer ν^f and ν^b. For the classes that



	Exact EM		Variational EM	
Class	ν_f	ν_b	ν_f	ν_b
1	0.04	0.11	0.06	0.07
2	0	0.04	0.07	0.12
3	0.20	0	0.07	0.04
4	0	0.14	0.07	0.06
5	0.19	0	0.07	0.15
6	0.17	0	0.09	0.07
7	0.19	0	0.10	0.03
8	0	0.13	0.03	0.08
9	0	0.13	0.09	0.06
10	0.21	0	0.02	0.03
11	0	0.12	0.10	0.04
12	0	0.17	0.04	0.06
13	0	0	0.07	0.15
14	0	0.16	0.12	0.04

Figure 8: Comparison of the learned parameters of the model in Sec. 2.1 using various learning methods. For each method, we show the mask probabilities α_k, pixel means μ_k, and pixel variances ψ_k for each class k as images, where black indicates a variance of 0. For exact EM and variational EM, we also show the total posterior probability that each class is used in modeling the foreground (ν^f) and background (ν^b): $\nu_k^f = \frac{1}{T}\sum_t Q(f^{(t)} = k)$, $\nu_k^b = \frac{1}{T}\sum_t Q(b^{(t)} = k)$. These indicate when a class accounts for too much or too little data. Note that there is no reason for the same class index for two techniques to correspond to the same object.

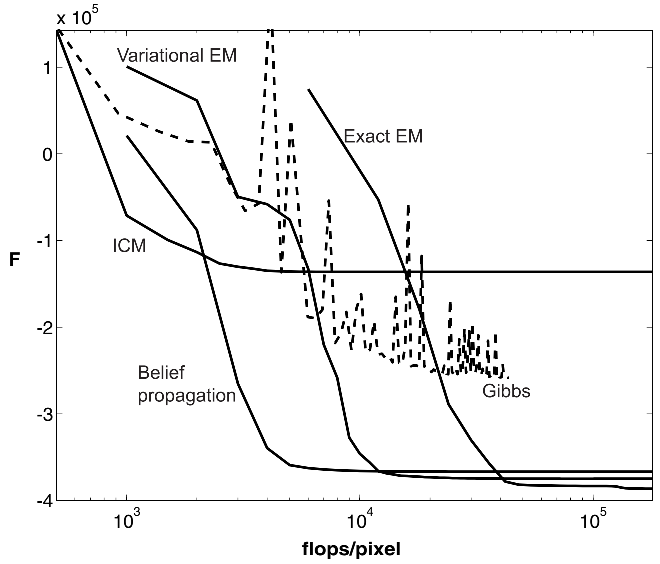


Figure 9: Free energy versus number of floating point operations used during training, for ICM, exact EM, and EM using Gibbs sampling, variational inference, and the sum-product algorithm in the E step.

exact EM always inferred as background classes, the variational technique learned masks probabilities that allow cutting holes in various places in order to place the classes in the foreground and show the faces

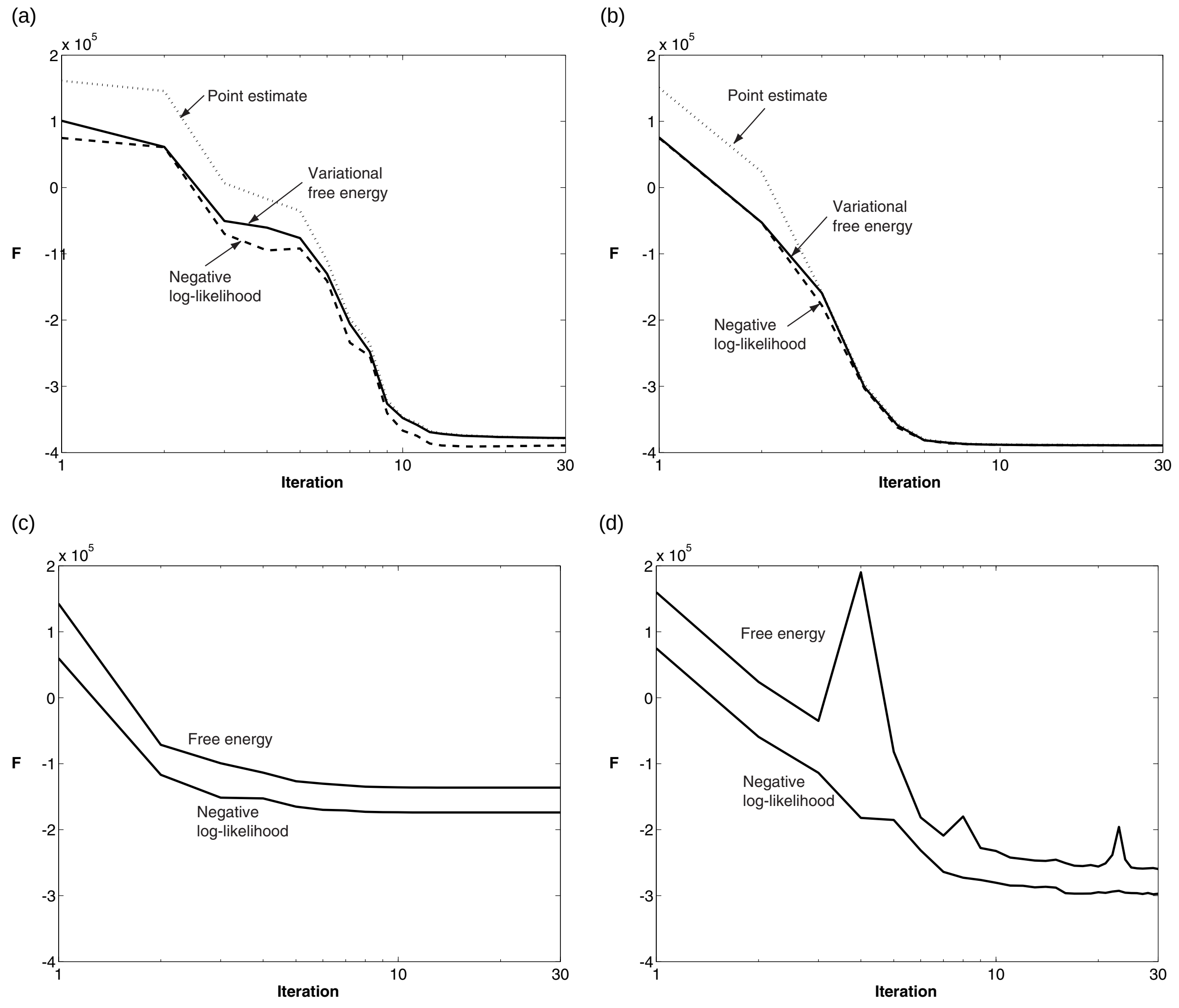


Figure 10: How good are the free energy approximations to the negative log-likelihood? In (a) we compare the mean-field variational free energy, the point estimate free energy and the negative log-likelihood during *variational EM*. In (b) we compare the same three quantities during *exact EM*. To further illustrate the advantage of modeling uncertainty in the posterior, in (c), we show the point-estimate free energy and the negative log-likelihood during *ICM learning*. In (d), we compare the same two quantities during *Gibbs sampling EM*.

behind them. The mask probabilities for these classes show outlines of faces and have values that are between zero and one indicating that the corresponding pixels are not consistently used when the class is picked to be in the foreground. Such mask values reduce the overall likelihood of the data and increase the variational free energy, because the mask likelihood $P(m_i|f) = \alpha_{fi}^{m_i}(1 - \alpha_{fi})^{1-m_i}$ has the highest value when α_{fi} is either 0 or 1, and m_i has the same value. Consequently,the variational free energy is always somewhat above the negative likelihood of the data for any given parameters (see Fig. 10a). Similar behavior is evident in the results of other approximate learning techniques that effectively decouple the posterior over the foreground and background classes, such as loopy belief propagation (last column of Fig. 8), and the structured variational technique. Note that small differences in free energy may or may not indicate a difference in the visual quality of the solution.

One concern that is sometimes raised about minimizing the free energy, is that the approximate Q-distribution used for the hidden RVs may not be well-suited to the model, causing the free energy to be a poor bound on the negative log-likelihood. However, as pointed out in [18], since the free energy is $F(Q, P) = D(Q, P) - \ln P(v)$ (see (6)), if two models fit the data equally well ($\ln P(v)$ is the same), minimizing the free energy will select the model that makes the approximate Q-distribution more exact (select P to minimize $D(Q, P)$).

We see this effect experimentally in Fig. 10. In Fig. 10a we show the free energy for the variational mean-field method during 30 iterations of learning. In this case, a single iteration corresponds to the shortest sequence of steps that update all variational parameters ($Q(b), Q(f), Q(m_i)$ for each training case) and all model parameters. In the same plot, we show the true negative log-likelihood after each iteration. We also show the point estimate of the free energy, which is evaluated at the modes of the variational posterior. Since the parameters are updated using the variational technique, the variational bound is the only one of the curves that theoretically has to be monotonic. While the negative of the log-likelihood is consistently better than the other estimates, the bound *does* appear to be relatively tight most of the time. Note that early on in learning, the point estimate gives a poor bound, but after learning is essentially finished, the point estimate gives a good bound. The fact that ICM performs poorly for learning, but performs well for inference after learning using a better technique, indicates the importance of accounting for uncertainty *early* in the learning process.

As shown in Fig. 10b, if the same energies are plotted for the parameters after each iteration of *exact* EM, the curves converge by the 5th iteration. Here, the mean-field variational free energy is computed using the factorized posterior $Q(f)Q(b)\prod_i Q(m_i)$ fitted by minimizing the KL distance to the exact posterior $P(f, b, m|z)$, while the point estimate is computed by further discarding everything but the peaks in the variational posterior. When the posterior is still broad early in the learning process, the variational posterior leads to a tighter bound on the negative log-likelihood than the point estimate. However, the point estimate catches up quickly as EM converges and the true posterior itself becomes peaked.

If the parameters are updated using ICM (which uses point estimates), as shown in Fig. 10c, poor local minima are found and both the free energy and the true negative log-likelihood are significantly worse than the same quantities found using exact EM and variational EM. Also, even after convergence, the point estimate free energy is not a tight bound on the negative log-likelihood.

These plots are meant to illustrate that while fairly severe approximations of the posterior can provide a tight bound near the local optimum of the log-likelihood, it is the behavior of the learning algorithm in the early iterations that determines how close an approximate technique will get to a local optimum of the the true log-likelihood. In the early iterations, to give the model a chance to get to a good local optimum, the model parameters are typically initialized to model broad distributions, allowing the learning techniques to explore more broadly the space of possibilities through relatively flat posteriors (*e.g.*, in our case we initialize the variances to be equal to one, corresponding to a standard deviation of 100% of the dynamic range of the image). If the approximate posterior makes greedy decisions early in the learning process, it is often difficult to correct the errors in later iterations. ICM, while very fast, is the most greedy of all the techniques. Even if variances are initialized to large values, ICM makes poor, greedy decisions for the configuration of the hidden RVs early on in learning, and does not recover from these mistakes.

Importantly, even computationally simple ways of accounting for uncertainty can improve performance significantly, in comparison with ICM. In Fig. 10d, we show the point estimate free energy and the negative log-likelihood when the ICM technique is modified to take some uncertainty into account, by performing a Gibbs sampling step for each RV, instead of picking the most probable value[5]. This method does not increase the computational cost per iteration compared to ICM, but it obtains much better values of both energies. Sampling sometimes makes the free energy worse during the learning, but allows the algorithm to account for uncertainty early on, when the true posterior distributions for RVs are broad. While this single-step Gibbs sampling technique obtains better energies than ICM, it does not achieve the lower energies obtained by exact EM and variational EM.

The effect of approximate probabilistic inference on the visual quality of the parameters is illustrated in Fig. 11, where we show how the model parameters change during several iterations of EM where the E step is performed using the sum-product algorithm. On the far right of the figure, we illustrate the inference over hidden RVs (foreground class f, background class b and the mask m) for 2 training cases. After the first iteration, while finding good guesses for the classes that took part in the formation process, the foreground and background are incorrectly inverted in the posterior for the first training case, and this situation persists even after convergence. Interestingly, by applying an additional 2 iterations of exact EM after 30 iterations of sum-product EM, the model leaves the local minimum. This is evident not only in the first training case, but also in the rest of the training data, as evidenced by the erasure of holes in the estimated mask probabilities for the background classes. The same improvement can be observed for the variational technique. In fact, adding a small number of exact EM iterations to improve the results of variational learning can be seen as part of the same framework of optimizing the variational free energy, except that not only the parameters of the variational posterior, but also its form can be varied to increase

[5]Note that because this technique does not use an ensemble of samples, it is not guaranteed to minimize free energy at each step.

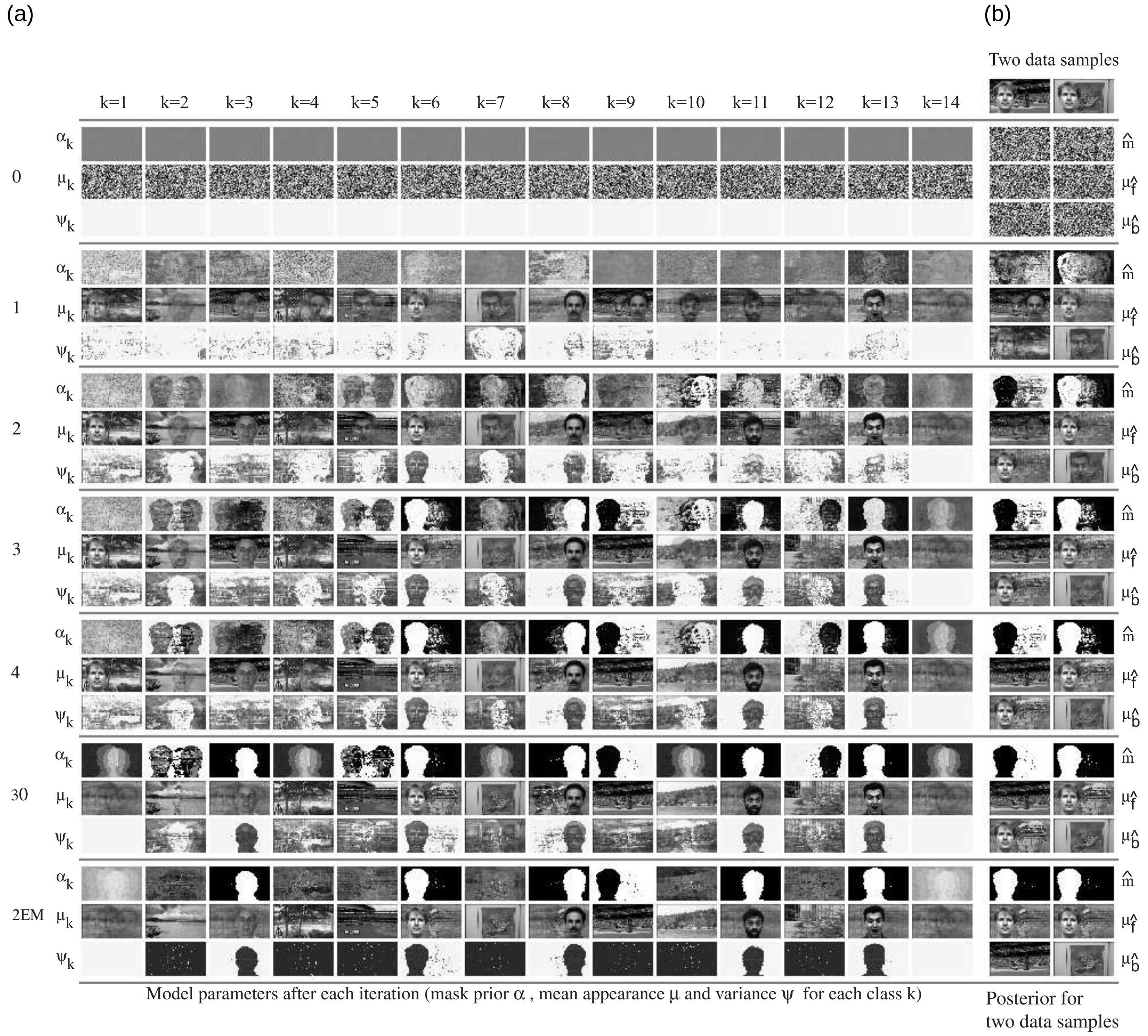


Figure 11: An illustration of learning using loopy belief propagation (the sum-product algorithm). For each iteration, we show: (a) model parameters, including mask priors, mean and variance parameters for each class, and (b) inferred distribution over the mask and the most likely foreground and background class for two of the 300 training cases. Although the algorithm (Sec. 5.14) converges quickly, it cannot escape a local minimum caused by an overly-greedy decision made in the very first iteration, in which the foreground object is placed into the background layer for the first illustrated training case. In this local minimum, some "background classes" (*e.g.*, $k = 12$) are used as foregrounds (see the mask). An additional 2 iterations of exact EM (Sec. 5.9), which uses the exact posterior $Q(f, b)Q(m|f, b)$, allows the inference process to flip the foreground and background where needed, and escape the local minimum (see the mask of class $k = 12$ after EM).

the bound at each step.

When the nature of the local minima to which a learning technique is susceptible is well understood, it is often possible to change either the model or the form of the approximation to the posterior, to avoid these minima without too much extra computation. In the occlusion model, the problem is the background-foreground inversion, which can be avoided by simply testing the inversion hypothesis and switching the inferred background and foreground classes to check if this lowers the free energy, rather than exploring all possible combinations of classes in the exact posterior. An elegant way of doing this within the variational framework is to add an additional “switch” RV to the model, which in the generative process can switch the two classes. Then, the mean field posterior would have a component that models the uncertainty about foreground-background inversion. While this would render the variational learning two times slower, it would still be much faster than the exact EM.

8 Future Directions

In our view, the most interesting and potentially high-impact areas of current research include introducing effective representations and models of data; inventing new inference and learning algorithms, that can efficiently infer combinatorial explanations of data; developing real-time, or near-real-time, modular software systems that enable researchers and developers to evaluate the effectiveness of combinations of inference and learning algorithms for solving real-world tasks; advancing techniques for combining information from multiple sources, *e.g.*, camera images, spectral features, microphones, text, tactile information, *etc.*; developing inference algorithms for active tasks, that effectively account for uncertainties in the sensory inputs and the model of the environment, when making decisions about investigating the environment. In our view, a core requirement in all of these directions of research is that uncertainty should be properly accounted for, both in the representations of problems and in adapting to new data. Large-scale, hierarchical probability models and efficient inference and learning algorithms will play a large role in the successful implementation of these systems.

References

[1] E. H. Adelson and P. Anandan. Ordinal characteristics of transparency. In *Proceedings of AAAI Workshop on Qualitative Vision*, 1990.

[2] O. E. Barndorff-Nielson. *Information and Exponential Families*. Wiley Chichester, 1978.

[3] J. Besag. On the statistical analysis of dirty pictures. *Journal of the Royal Statistical Society B*, 48:259–302, 1986.

[4] R. G. Cowell, A. P. Dawid, S. L. Lauritzen, and D. J. Spiegelhalter. *Probabilistic Networks and Expert Systems*. Springer, New York NY., 1999.

[5] R. G. Cowell, A. P. Dawid, and P. Sebastiani. A comparison of sequential learning methods for incomplete data. *Journal of Bayesian Statistics*, 5:581–588, 1996.

[6] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Proceedings of the Royal Statistical Society*, B-39:1–38, 1977.

[7] D. Cahan (editor). *Hermann von Helmholtz*. University of California Press, Los Angeles CA., 1993.

[8] W. Freeman and E. Pasztor. Learning low-level vision. In *Proceedings of the International Conference on Computer Vision*, pages 1182–1189, 1999.

[9] B. J. Frey. Extending factor graphs so as to unify directed and undirected graphical models. In *Proceedings of the Nineteenth Conference on Uncertainty in Artificial Intelligence*. Morgan Kaufmann, Acapulco, Mexico, 2003.

[10] B. J. Frey and N. Jojic. Transformed component analysis: Joint estimation of spatial transformations and image components. In *Proceedings of the IEEE International Conference on Computer Vision*, September 1999.

[11] B. J. Frey and N. Jojic. Transformation-invariant clustering using the EM algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(1), 2003.

[12] B. J. Frey, N. Jojic, and A. Kannan. Learning appearance and transparency manifolds of occluded objects in layers. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2003.

[13] B. J. Frey, R. Koetter, and N. Petrovic. Very loopy belief propagation for unwrapping phase images. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems 14*. MIT Press, 2002.

[14] B. J. Frey and D. J. C. MacKay. A revolution: Belief propagation in graphs with cycles. In M. I. Jordan, M. I. Kearns, and S. A. Solla, editors, *Advances in Neural Information Processing Systems 1997, Volume 10*, pages 479–485. MIT Press, 1998.

[15] S. Geman and D. Geman. Stochastic relaxation, Gibbs distribution and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6:721–741, 1984.

[16] Z. Ghahramani and M. Beal. Propagation algorithms for variational Bayesian learning. In T. Leen, T. Dietterich, and V. Tresp, editors, *Advances in Neural Information Processing Systems 13*. MIT Press, 2001.

[17] D. Heckerman. A tutorial on learning with Bayesian networks. In M. I. Jordan, editor, *Learning in Graphical Models*. Kluwer Academic Publishers, Norwell MA., 1998.

[18] G. E. Hinton, P. Dayan, B. J. Frey, and R. M. Neal. The wake-sleep algorithm for unsupervised neural networks. *Science*, 268:1158–1161, 1995.

[19] G. E. Hinton and T. J. Sejnowski. Learning and relearning in Boltzmann machines. In D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume I, pages 282–317. MIT Press, Cambridge MA., 1986.

[20] N. Jojic and B. J. Frey. Learning flexible sprites in video layers. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2001.

[21] N. Jojic, B. J. Frey, and A. Kannan. Epitomic analysis of appearance and shape. In *Proceedings of the IEEE International Conference on Computer Vision*, September 2003.

[22] N. Jojic, N. Petrovic, B. J. Frey, and T. S. Huang. Transformed hidden markov models: Estimating mixture models of images and inferring spatial transformations in video sequences. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, June 2000.

[23] N. Jojic, P. Simard, B. J. Frey, and D. Heckerman. Separating appearance from deformation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, July 2001.

[24] M. I. Jordan, Z. Ghahramani, T. S. Jaakkola, and L. K. Saul. An introduction to variational methods for graphical models. In M. I. Jordan, editor, *Learning in Graphical Models*. Kluwer Academic Publishers, Norwell MA., 1998.

[25] F. R. Kschischang, B. J. Frey, and H.-A. Loeliger. Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory, Special Issue on Codes on Graphs and Iterative Algorithms*, 47(2):498–519, February 2001.

[26] S. L. Lauritzen. *Graphical Models*. Oxford University Press, New York NY., 1996.

[27] D. J. C. MacKay. Bayesian neural networks and density networks. *Nuclear Instruments and Methods in Physics Research*, 354:73–80, 1995.

[28] M. Mézard, G. Parisi, and R. Zecchina. Analytic and algorithmic solution of random satisfiability problems. *Science*, 297:812–815, 2002.

[29] T. P. Minka. Expectation propagation for approximate Bayesian inference. In *Uncertainty in Artificial Intelligence 2001*. Morgan Kaufmann, Seattle, Washington, 2001.

[30] Kevin P. Murphy, Yair Weiss, and Michael I. Jordan. Loopy belief propagation for approximate inference: An empirical study. In *Uncertainty in Artificial Intelligence 1999*. Stockholm, Sweden, 1999.

[31] R. M. Neal. Bayesian mixture modeling by Monte Carlo simulation. Technical Report CRG-TR-91-2, University of Toronto, 1991.

[32] R. M. Neal. Probabilistic inference using Markov chain Monte Carlo methods. University of Toronto Technical Report, 1993.

[33] R. M. Neal and G. E. Hinton. A view of the EM algorithm that justifies incremental, sparse, and other variants. In M. I. Jordan, editor, *Learning in Graphical Models*, pages 355–368. Kluwer Academic Publishers, Norwell MA., 1998.

[34] A. Y. Ng and M. I. Jordan. A comparison of logistic regression and naive Bayes. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems 14*. MIT Press, Cambridge MA., 2002.

[35] J. Pearl. *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann, San Mateo CA., 1988.

[36] M. J. Wainwright and M. I. Jordan. Graphical models, variational inference and exponential families. Technical Report 649, UC Berkeley, Dept. of Statistics, 2003.

[37] Y. Weiss and W. Freeman. On the optimaility of solutions of the max-product belief propagation algorithm in arbitrary graphs. *IEEE Transactions on Information Theory, Special Issue on Codes on Graphs and Iterative Algorithms*, 47(2):736–744, February 2001.

[38] J. Yedidia, W. T. Freeman, and Y. Weiss. Understanding belief propagation and its generalizations. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 2001.