Unsupervised Learning

Zoubin Ghahramani*

Gatsby Computational Neuroscience Unit, University College London, UK
zoubin@gatsby.ucl.ac.uk
http://www.gatsby.ucl.ac.uk/~zoubin

Abstract. We give a tutorial and overview of the field of unsupervised learning from the perspective of statistical modeling. Unsupervised learning can be motivated from information theoretic and Bayesian principles. We briefly review basic models in unsupervised learning, including factor analysis, PCA, mixtures of Gaussians, ICA, hidden Markov models, state-space models, and many variants and extensions. We derive the EM algorithm and give an overview of fundamental concepts in graphical models, and inference algorithms on graphs. This is followed by a quick tour of approximate Bayesian inference, including Markov chain Monte Carlo (MCMC), Laplace approximation, BIC, variational approximations, and expectation propagation (EP). The aim of this chapter is to provide a high-level view of the field. Along the way, many state-of-the-art ideas and future directions are also reviewed.

1 Introduction

Machine learning is the field of research devoted to the formal study of learning systems. This is a highly interdisciplinary field which borrows and builds upon ideas from statistics, computer science, engineering, cognitive science, optimization theory and many other disciplines of science and mathematics. The purpose of this chapter is to introduce in a fairly concise manner the key ideas underlying the sub-field of machine learning known as *unsupervised learning*. This introduction is necessarily incomplete given the enormous range of topics under the rubric of unsupervised learning. The hope is that interested readers can delve more deeply into the many topics covered here by following some of the cited references. The chapter starts at a highly tutorial level but will touch upon state-of-the-art research in later sections. It is assumed that the reader is familiar with elementary linear algebra, probability theory, and calculus, but not much else.

1.1 What Is Unsupervised Learning?

Consider a machine (or living organism) which receives some sequence of inputs $x_1, x_2, x_3, \ldots$, where x_t is the sensory input at time t. This input, which we will

* The author is also at the Center for Automated Learning and Discovery, Carnegie Mellon University, USA.

O. Bousquet et al. (Eds.): Machine Learning 2003, LNAI 3176, pp. 72–112, 2004.

often call the *data*, could correspond to an image on the retina, the pixels in a camera, or a sound waveform. It could also correspond to less obviously sensory data, for example the words in a news story, or the list of items in a supermarket shopping basket.

One can distinguish between four different kinds of machine learning. In *supervised learning* the machine[1] is also given a sequence of desired outputs $y_1, y_2, \ldots$, and the goal of the machine is to learn to produce the correct output given a new input. This output could be a class label (in classification) or a real number (in regression).

In *reinforcement learning* the machine interacts with its environment by producing actions $a_1, a_2, \ldots$. These actions affect the state of the environment, which in turn results in the machine receiving some scalar rewards (or punishments) $r_1, r_2, \ldots$. The goal of the machine is to learn to act in a way that maximizes the future rewards it receives (or minimizes the punishments) over its lifetime. Reinforcement learning is closely related to the fields of decision theory (in statistics and management science), and control theory (in engineering). The fundamental problems studied in these fields are often formally equivalent, and the solutions are the same, although different aspects of problem and solution are usually emphasized.

A third kind of machine learning is closely related to *game theory* and generalizes reinforcement learning. Here again the machine gets inputs, produces actions, and receives rewards. However, the environment the machine interacts with is not some static world, but rather it can contain other machines which can also sense, act, receive rewards, and learn. Thus the goal of the machine is to act so as to maximize rewards in light of the other machines' current and future actions. Although there is a great deal of work in game theory for simple systems, the dynamic case with multiple adapting machines remains an active and challenging area of research.

Finally, in *unsupervised learning* the machine simply receives inputs $x_1, x_2, \ldots$, but obtains neither supervised target outputs, nor rewards from its environment. It may seem somewhat mysterious to imagine what the machine could possibly learn given that it doesn't get any feedback from its environment. However, it is possible to develop of formal framework for unsupervised learning based on the notion that the machine's goal is to build representations of the input that can be used for decision making, predicting future inputs, efficiently communicating the inputs to another machine, etc. In a sense, unsupervised learning can be thought of as finding patterns in the data above and beyond what would be considered pure unstructured noise. Two very simple classic examples of unsupervised learning are clustering and dimensionality reduction. We discuss these in Section 2. The remainder of this chapter focuses on unsupervised learning,

[1] Henceforth, for succinctness I'll use the term machine to refer both to machines and living organisms. Some people prefer to call this a system or agent. The same mathematical theory of learning applies regardless of what we choose to call the learner, whether it is artificial or biological.

although many of the concepts discussed can be applied to supervised learning as well. But first, let us consider how unsupervised learning relates to statistics and information theory.

1.2 Machine Learning, Statistics, and Information Theory

Almost all work in unsupervised learning can be viewed in terms of learning a probabilistic model of the data. Even when the machine is given no supervision or reward, it may make sense for the machine to estimate a model that represents the probability distribution for a new input x_t given previous inputs $x_1, \ldots, x_{t-1}$ (consider the obviously useful examples of stock prices, or the weather). That is, the learner models $P(x_t|x_1, \ldots, x_{t-1})$. In simpler cases where the order in which the inputs arrive is irrelevant or unknown, the machine can build a model of the data which assumes that the data points $x_1, x_2, \ldots$ are independently and identically drawn from some distribution $P(x)$[2].

Such a model can be used for *outlier detection* or *monitoring*. Let x represent patterns of sensor readings from a nuclear power plant and assume that $P(x)$ is learned from data collected from a normally functioning plant. This model can be used to evaluate the probability of a new sensor reading; if this probability is abnormally low, then either the model is poor or the plant is behaving abnormally, in which case one may want to shut it down.

A probabilistic model can also be used for *classification*. Assume $P_1(x)$ is a model of the attributes of credit card holders who paid on time, and $P_2(x)$ is a model learned from credit card holders who defaulted on their payments. By evaluating the relative probabilities $P_1(x')$ and $P_2(x')$ on a new applicant x', the machine can decide to classify her into one of these two categories.

With a probabilistic model one can also achieve efficient *communication* and *data compression*. Imagine that we want to transmit, over a digital communication line, symbols x randomly drawn from $P(x)$. For example, x may be letters of the alphabet, or images, and the communication line may be the Internet. Intuitively, we should encode our data so that symbols which occur more frequently have code words with fewer bits in them, otherwise we are wasting bandwidth. Shannon's source coding theorem quantifies this by telling us that the optimal number of bits to use to encode a symbol with probability $P(x)$ is $-\log_2 P(x)$. Using these number of bits for each symbol, the expected coding cost is the entropy of the distribution P.

$$H(P) \stackrel{\text{def}}{=} -\sum_x P(x) \log_2 P(x) \tag{1}$$

In general, the true distribution of the data is unknown, but we can learn a model of this distribution. Let's call this model $Q(x)$. The optimal code *with*

[2] We will use both P and p to denote probability distributions and probability densities. The meaning should be clear depending on whether the argument is discrete or continuous.

respect to this model would use $-\log_2 Q(x)$ bits for each symbol x. The expected coding cost, taking expectations with respect to the true distribution, is

$$-\sum_x P(x) \log_2 Q(x) \tag{2}$$

The difference between these two coding costs is called the Kullback-Leibler (KL) divergence

$$\mathrm{KL}(P\|Q) \stackrel{\mathrm{def}}{=} \sum_x P(x) \log \frac{P(x)}{Q(x)} \tag{3}$$

The KL divergence is non-negative and zero if and only if P=Q. It measures the coding inefficiency in bits from using a model Q to compress data when the true data distribution is P. *Therefore, the better our model of the data, the more efficiently we can compress and communicate new data.* This is an important link between machine learning, statistics, and information theory. An excellent text which elaborates on these relationships and many of the topics in this chapter is [1].

1.3 Bayes Rule

Bayes rule,

$$P(y|x) = \frac{P(x|y)P(y)}{P(x)} \tag{4}$$

which follows from the equality $P(x, y) = P(x)P(y|x) = P(y)P(x|y)$, can be used to motivate a coherent statistical framework for machine learning. The basic idea is the following. Imagine we wish to design a machine which has beliefs about the world, and updates these beliefs on the basis of observed data. The machine must somehow represent the strengths of its beliefs numerically. It has been shown that if you accept certain axioms of coherent inference, known as the *Cox axioms*, then a remarkable result follows [2]: If the machine is to represent the strength of its beliefs by real numbers, then the only reasonable and coherent way of manipulating these beliefs is to have them satisfy the rules of probability, such as Bayes rule. Therefore, $P(X = x)$ can be used not only to represent the frequency with which the variable X takes on the value x (as in so-called frequentist statistics) but it can also be used to represent the degree of belief that $X = x$. Similarly, $P(X = x|Y = y)$ can be used to represent the degree of belief that $X = x$ given that one knows $Y = y$.[3]

[3] Another way to motivate the use of the rules of probability to encode degrees of belief comes from game-theoretic arguments in the form of the *Dutch Book Theorem*. This theorem states that if you are willing to accept bets with odds based on your degrees of beliefs, then unless your beliefs are coherent in the sense that they satisfy the rules of probability theory, there exists a set of simultaneous bets (called a "Dutch Book") which you will accept and which is guaranteed to lose you money, no matter what the outcome. The only way to ensure that Dutch Books don't exist against you, is to have degrees of belief that satisfy Bayes rule and the other rules of probability theory.

From Bayes rule we derive the following simple framework for machine learning. Assume a universe of models Ω; let $\Omega = \{1, \ldots, M\}$ although it need not be finite or even countable. The machines starts with some prior beliefs over models $m \in \Omega$ (we will see many examples of models later), such that $\sum_{m=1}^{M} P(m) = 1$. A model is simply some probability distribution over data points, i.e. $P(x|m)$. For simplicity, let us further assume that in all the models the data is taken to be independently and identically distributed (i.i.d.). After observing a data set $\mathcal{D} = \{x_1, \ldots, x_N\}$, the beliefs over models is given by:

$$P(m|\mathcal{D}) = \frac{P(m)P(\mathcal{D}|m)}{P(D)} \propto P(m) \prod_{n=1}^{N} P(x_n|m) \tag{5}$$

which we read as the *posterior over models* is the *prior* multiplied by the *likelihood*, normalized.

The *predictive distribution* over new data, which would be used to encode new data efficiently, is

$$P(x|\mathcal{D}) = \sum_{m=1}^{M} P(x|m)P(m|\mathcal{D}) \tag{6}$$

Again this follows from the rules of probability theory, and the fact that the models are assumed to produce i.i.d. data.

Often models are defined by writing down a parametric probability distribution (again, we'll see many examples below). Thus, the model m might have parameters θ, which are assumed to be unknown (this could in general be a vector of parameters). To be a well-defined model from the perspective of Bayesian learning, one has to define a prior over these model parameters $P(\theta|m)$ which naturally has to satisfy the following equality

$$P(x|m) = \int P(x|\theta, m)P(\theta|m)d\theta \tag{7}$$

Given the model m it is also possible to infer the posterior over the parameters of the model, i.e. $P(\theta|\mathcal{D}, m)$, and to compute the predictive distribution, $P(x|\mathcal{D}, m)$. These quantities are derived in exact analogy to equations (5) and (6), except that instead of summing over possible models, we integrate over parameters of a particular model. All the key quantities in Bayesian machine learning follow directly from the basic rules of probability theory.

Certain approximate forms of Bayesian learning are worth mentioning. Let's focus on a particular model m with parameters θ, and an observed data set $\mathcal{D}$. The predictive distribution averages over all possible parameters weighted by the posterior

$$P(x|\mathcal{D}, m) = \int P(x|\theta)P(\theta|\mathcal{D}, m)d\theta. \tag{8}$$

In certain cases, it may be cumbersome to represent the entire posterior distribution over parameters, so instead we will choose to find a *point-estimate*

of the parameters $\hat{\theta}$. A natural choice is to pick the most probable parameter value given the data, which is known as the *maximum a posteriori* or MAP parameter estimate

$$\hat{\theta}_{\text{MAP}} = \arg\max_{\theta} P(\theta|\mathcal{D}, m) = \arg\max_{\theta} \left[\log P(\theta|m) + \sum_{n} \log P(x_n|\theta, m) \right] \quad (9)$$

Another natural choice is the *maximum likelihood* or ML parameter estimate

$$\hat{\theta}_{\text{ML}} = \arg\max_{\theta} P(\mathcal{D}|\theta, m) = \arg\max_{\theta} \sum_{n} \log P(x_n|\theta, m) \quad (10)$$

Many learning algorithms can be seen as finding ML parameter estimates. The ML parameter estimate is also acceptable from a frequentist statistical modeling perspective since it does not require deciding on a prior over parameters. However, ML estimation does not protect against overfitting—more complex models will generally have higher maxima of the likelihood. In order to avoid problems with overfitting, frequentist procedures often maximize a *penalized* or *regularized* log likelihood (e.g. [3]). If the penalty or regularization term is interpreted as a log prior, then maximizing penalized likelihood appears identical to maximizing a posterior. However, there are subtle issues that make a Bayesian MAP procedure and maximum penalized likelihood different [4]. One difference is that the MAP estimate is not invariant to reparameterization, while the maximum of the penalized likelihood is invariant. The penalized likelihood is a function, not a density, and therefore does not increase or decrease depending on the Jacobian of the reparameterization.

2 Latent Variable Models

The framework described above can be applied to a wide range of models. No singe model is appropriate for all data sets. The art in machine learning is to develop models which are appropriate for the data set being analyzed, and which have certain desired properties. For example, for high dimensional data sets it might be necessary to use models that perform dimensionality reduction. Of course, ultimately, the machine should be able to decide on the appropriate model without any human intervention, but to achieve this in full generality requires significant advances in artificial intelligence.

In this section, we will consider probabilistic models that are defined in terms of some latent or hidden variables. These models can be used to do dimensionality reduction and clustering, the two cornerstones of unsupervised learning.

2.1 Factor Analysis

Let the data set $\mathcal{D}$ consist of D-dimensional real valued vectors, $\mathcal{D} = \{\mathbf{y}_1, \ldots, \mathbf{y}_N\}$. In factor analysis, the data is assumed to be generated from the following model

$$\mathbf{y} = \Lambda \mathbf{x} + \epsilon \quad (11)$$

where $\mathbf{x}$ is a K-dimensional zero-mean unit-variance multivariate Gaussian vector with elements corresponding to hidden (or latent) factors, Λ is a $D \times K$ matrix of parameters, known as the factor loading matrix, and ϵ is a D-dimensional zero-mean multivariate Gaussian noise vector with diagonal covariance matrix Ψ. Defining the parameters of the model to be $\theta = (\Psi, \Lambda)$, by integrating out the factors, one can readily derive that

$$p(\mathbf{y}|\theta) = \int p(\mathbf{x}|\theta)p(\mathbf{y}|\mathbf{x},\theta)d\mathbf{x} = \mathcal{N}(0, \Lambda\Lambda^\top + \Psi) \tag{12}$$

where $\mathcal{N}(\mu, \Sigma)$ refers to a multivariate Gaussian density with mean μ and covariance matrix Σ. For more details refer to [5].

Factor analysis is an interesting model for several reasons. If the data is very high dimensional (D is large) then even a simple model like the full-covariance multivariate Gaussian will have too many parameters to reliably estimate or infer from the data. By choosing $K < D$, factor analysis makes it possible to model a Gaussian density for high dimensional data without requiring $\mathcal{O}(D^2)$ parameters. Moreover, given a new data point, one can compute the posterior over the hidden factors, $p(\mathbf{x}|\mathbf{y},\theta)$; since $\mathbf{x}$ is lower dimensional than $\mathbf{y}$ this provides a low-dimensional representation of the data (for example, one could pick the mean of $p(\mathbf{x}|\mathbf{y},\theta)$ as the representation for $\mathbf{y}$).

2.2 Principal Components Analysis (PCA)

Principal components analysis (PCA) is an important limiting case of factor analysis (FA). One can derive PCA by making two modifications to FA. First, the noise is assumed to be isotropic, in other words each element of ϵ has equal variance: $\Psi = \sigma^2 I$, where I is a $D \times D$ identity matrix. This model is called *probabilistic PCA* [6, 7]. Second, if we take the limit of $\sigma \to 0$ in probabilistic PCA, we obtain standard PCA (which also goes by the names Karhunen-Loève expansion, and singular value decomposition; SVD). Given a data set with covariance matrix Σ, for maximum likelihood factor analysis the goal is to find parameters Λ, and Ψ for which the model $\Lambda\Lambda^\top + \Psi$ has highest likelihood. In PCA, the goal is to find Λ so that the likelihood is highest for $\Lambda\Lambda^\top$. Note that this matrix is singular unless $K = D$, so the standard PCA model is not a sensible model. However, taking the limiting case, and further constraining the columns of Λ to be orthogonal, it can be derived that the principal components correspond to the K eigenvectors with largest eigenvalue of Σ. PCA is thus attractive because the solution can be found immediately after eigendecomposition of the covariance. Taking the limit $\sigma \to 0$ of $p(\mathbf{x}|\mathbf{y},\Lambda,\sigma)$ we find that it is a delta-function at $\mathbf{x} = \Lambda^\top\mathbf{y}$, which is the projection of $\mathbf{y}$ onto the principal components.

2.3 Independent Components Analysis (ICA)

Independent components analysis (ICA) extends factor analysis to the case where the factors are non-Gaussian. This is an interesting extension because

many real-world data sets have structure which can be modeled as linear combinations of sparse sources. This includes auditory data, images, biological signals such as EEG, etc. *Sparsity* simply corresponds to the assumption that the factors have distributions with higher kurtosis that the Gaussian. For example, $p(x) = \frac{\lambda}{2}\exp\{-\lambda|x|\}$ has a higher peak at zero and heavier tails than a Gaussian with corresponding mean and variance, so it would be considered sparse (strictly speaking, one would like a distribution which had non-zero probability mass at 0 to get true sparsity).

Models like PCA, FA and ICA can all be implemented using neural networks (multi-layer perceptrons) trained using various cost functions. It is not clear what advantage this implementation/interpretation has from a machine learning perspective, although it provides interesting ties to biological information processing.

Rather than ML estimation, one can also do Bayesian inference for the parameters of probabilistic PCA, FA, and ICA.

2.4 Mixture of Gaussians

The densities modeled by PCA, FA and ICA are all relatively simple in that they are unimodal and have fairly restricted parametric forms (Gaussian, in the case of PCA and FA). To model data with more complex structure such as clusters, it is very useful to consider mixture models. Although it is straightforward to consider mixtures of arbitrary densities, we will focus on Gaussians as a common special case. The density of each data point in a mixture model can be written:

$$p(\mathbf{y}|\theta) = \sum_{k=1}^{K} \pi_k \, p(\mathbf{y}|\theta_k) \tag{13}$$

where each of the K components of the mixture is, for example, a Gaussian with differing means and covariances $\theta_k = (\mu_k, \Sigma_k)$ and π_k is the mixing proportion for component k, such that $\sum_{k=1}^{K} \pi_k = 1$ and $\pi_k > 0, \quad \forall k$.

A different way to think about mixture models is to consider them as latent variable models, where associated with each data point is a K-ary discrete latent (i.e. hidden) variable s which has the interpretation that $s = k$ if the data point was generated by component k. This can be written

$$p(\mathbf{y}|\theta) = \sum_{k=1}^{K} P(s = k|\pi) p(\mathbf{y}|s = k, \theta) \tag{14}$$

where $P(s = k|\pi) = \pi_k$ is the prior for the latent variable taking on value k, and $p(\mathbf{y}|s = k, \theta) = p(\mathbf{y}|\theta_k)$ is the density under component k, recovering Equation (13).

2.5 K-Means

The mixture of Gaussians model is closely related to an unsupervised clustering algorithm known as k-means as follows: Consider the special case where all the

Gaussians have common covariance matrix proportional to the identity matrix: $\Sigma_k = \sigma^2 I$, $\forall k$, and let $\pi_k = 1/K$, $\forall k$. We can estimate the maximum likelihood parameters of this model using the iterative algorithm which we are about to describe, known as EM. The resulting algorithm, as we take the limit $\sigma^2 \to 0$, becomes exactly the k-means algorithm. Clearly the model underlying k-means has only singular Gaussians and is therefore an unreasonable model of the data; however, k-means is usually justified from the point of view of clustering to minimize a distortion measure, rather than fitting a probabilistic models.

3 The EM Algorithm

The EM algorithm is an algorithm for estimating ML parameters of a model with latent variables. Consider a model with observed variables $\mathbf{y}$, hidden/latent variables $\mathbf{x}$, and parameters θ. We can lower bound the log likelihood for any data point as follows

$$L(\theta) = \log p(\mathbf{y}|\theta) = \log \int p(\mathbf{x}, \mathbf{y}|\theta)\, d\mathbf{x} \tag{15}$$

$$= \log \int q(\mathbf{x}) \frac{p(\mathbf{x}, \mathbf{y}|\theta)}{q(\mathbf{x})}\, d\mathbf{x} \tag{16}$$

$$\geq \int q(\mathbf{x}) \log \frac{p(\mathbf{x}, \mathbf{y}|\theta)}{q(\mathbf{x})}\, d\mathbf{x} \stackrel{\text{def}}{=} F(q, \theta) \tag{17}$$

where $q(\mathbf{x})$ is some arbitrary density over the hidden variables, and the lower bound holds due to the concavity of the log function (this inequality is known as Jensen's inequality). The lower bound F is a functional of both the density $q(\mathbf{x})$ and the model parameters θ. For a data set of N data points $\mathbf{y}^{(1)}, \ldots, \mathbf{y}^{(N)}$, this lower bound is formed for the log likelihood term corresponding to each data point, thus there is a separate density $q^{(n)}(\mathbf{x})$ for each point and $F(q, \theta) = \sum_n F^{(n)}(q^{(n)}, \theta)$.

The basic idea of the Expectation-Maximization (EM) algorithm is to iterate between optimizing this lower bound as a function of q and as a function of θ. We can prove that this will never decrease the log likelihood. After initializing the parameters somehow, the k^{th} iteration of the algorithm consists of the following two steps:

E Step: *Optimize F with respect to the distribution q while holding the parameters fixed*

$$q_k(\mathbf{x}) = \arg\max_{q(\mathbf{x})} \int q(\mathbf{x}) \log \frac{p(\mathbf{x}, \mathbf{y}|\theta_{k-1})}{q(\mathbf{x})} \tag{18}$$

$$q_k(\mathbf{x}) = p(\mathbf{x}|\mathbf{y}, \theta_{k-1}) \tag{19}$$

M Step: *Optimize F with respect to the parameters θ while holding the distribution over hidden variables fixed*

$$\theta_k = \arg\max_\theta \int q_k(\mathbf{x}) \log \frac{p(\mathbf{x}, \mathbf{y}|\theta)}{q_k(\mathbf{x})} \, d\mathbf{x} \tag{20}$$

$$\theta_k = \arg\max_\theta \int q_k(\mathbf{x}) \log p(\mathbf{x}, \mathbf{y}|\theta) \, d\mathbf{x} \tag{21}$$

Let us be absolutely clear what happens for a data set of N data points: In the E step, for each data point, the distribution over the hidden variables is set to the posterior for that data point $q_k^{(n)}(\mathbf{x}) = p(\mathbf{x}|\mathbf{y}^{(n)}, \theta_{k-1})$, $\forall n$. In the M step the single set of parameters is re-estimated by maximizing the sum of the expected log likelihoods: $\theta_k = \arg\max_\theta \sum_n \int q_k^{(n)}(\mathbf{x}) \log p(\mathbf{x}, \mathbf{y}^{(n)}|\theta) \, d\mathbf{x}$.

Two things are still unclear: how does (19) follow from (18), and how is this algorithm guaranteed to increase the likelihood? The optimization in (18) can be written as follows since $p(\mathbf{x}, \mathbf{y}|\theta_{k-1}) = p(\mathbf{y}|\theta_{k-1})p(\mathbf{x}|\mathbf{y}, \theta_{k-1})$:

$$q_k(\mathbf{x}) = \arg\max_{q(\mathbf{x})} \Big[\log p(\mathbf{y}|\theta_{k-1}) + \int q(\mathbf{x}) \log \frac{p(\mathbf{x}|\mathbf{y}, \theta_{k-1})}{q(\mathbf{x})} \, d\mathbf{x} \Big] \tag{22}$$

Now, the first term is a constant w.r.t. $q(\mathbf{x})$ and the second term is the negative of the Kullback-Leibler divergence

$$\mathrm{KL}(q(\mathbf{x}) \| p(\mathbf{x}|\mathbf{y}, \theta_{k-1})) = \int q(\mathbf{x}) \log \frac{q(\mathbf{x})}{p(\mathbf{x}|\mathbf{y}, \theta_{k-1})} \, d\mathbf{x} \tag{23}$$

which we have seen in Equation (3) in its discrete form. This is minimized at $q(\mathbf{x}) = p(\mathbf{x}|\mathbf{y}, \theta_{k-1})$, where the KL divergence is zero. Intuitively, the interpretation of this is that in the E step of EM, the goal is to find the posterior distribution of the hidden variables given the observed variables and the current settings of the parameters. We also see that since the KL divergence is zero, at the end of the E step, $F(q_k, \theta_{k-1}) = L(\theta_{k-1})$.

In the M step, F is increased with respect to θ. Therefore, $F(q_k, \theta_k) \geq F(q_k, \theta_{k-1})$. Moreover, $L(\theta_k) = F(q_{k+1}, \theta_k) \geq F(q_k, \theta_k)$ after the next E step. We can put these steps together to establish that $L(\theta_k) \geq L(\theta_{k-1})$, establishing that the algorithm is guaranteed to increase the likelihood or keep it fixed (at convergence).

The EM algorithm can be applied to all the latent variable models described above, i.e. FA, probabilistic PCA, mixture models, and ICA. In the case of mixture models, the hidden variable is the discrete assignment s of data points to clusters; consequently the integrals turn into sums where appropriate. EM has wide applicability to latent variable models, although it is not always the fastest optimization method [8]. Moreover, we should note that the likelihood often has many local optima and EM will converge some local optimum which may not be the global one.

EM can also be used to estimate MAP parameters of a model, and as we will see in Section 11.4 there is a Bayesian generalization of EM as well.

4 Modeling Time Series and Other Structured Data

So far we have assumed that the data is *unstructured*, that is, the observations are assumed to be independent and identically distributed. This assumption is unreasonable for many data sets in which the observations arrive in a sequence and subsequent observations are correlated. Sequential data can occur in time series modeling (as in financial data or the weather) and also in situations where the sequential nature of the data is not necessarily tied to time (as in protein data which consist of sequences of amino acids).

As the most basic level, time series modeling consists of building a probabilistic model of the present observation given all past observations $p(\mathbf{y}_t|\mathbf{y}_{t-1}, \mathbf{y}_{t-2} \ldots)$. Because the history of observations grows arbitrarily large it is necessary to limit the complexity of such a model. There are essentially two ways of doing this.

The first approach is to limit the window of past observations. Thus one can simply model $p(\mathbf{y}_t|\mathbf{y}_{t-1})$ and assume that this relation holds for all t. This is known as a first-order Markov model. A second-order Markov model would be $p(\mathbf{y}_t|\mathbf{y}_{t-1}, \mathbf{y}_{t-2})$, and so on. Such Markov models have two limitations: First, the influence of past observations on present observations vanishes outside this window, which can be unrealistic. Second, it may be unnatural and unwieldy to model directly the relationship between raw observations at one time step and raw observations at a subsequent time step. For example, if the observations are noisy images, it would make more sense to de-noise them, extract some description of the objects, motions, illuminations, and then try to predict from that.

The second approach is to make use of latent or hidden variables. Instead of modeling directly the effect of $\mathbf{y}_{t-1}$ on $\mathbf{y}_t$, we assume that the observations were generated from some underlying hidden variable $\mathbf{x}_t$ which captures the dynamics of the system. For example, $\mathbf{y}$ might be noisy sonar readings of objects in a room, while $\mathbf{x}$ might be the actual locations and sizes of these objects. We usually call this hidden variable $\mathbf{x}$ the *state variable* since it is meant to capture all the aspects of the system relevant to predicting the future dynamical behavior of the system.

In order to understand more complex time series models, it is essential that one be familiar with state-space models (SSMs) and hidden Markov models (HMMs). These two classes of models have played a historically important role in control engineering, visual tracking, speech recognition, protein sequence modeling, and error decoding. They form the simplest building blocks from which other richer time-series models can be developed, in a manner completely analogous to the role that FA and mixture models play in building more complex models for i.i.d. data.

4.1 State-Space Models (SSMs)

In a state-space model, the sequence of observed data $\mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3, \ldots$ is assumed to have been generated from some sequence of hidden state variables $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \ldots$.

Letting $\mathbf{x}_{1:T}$ denote the sequence $\mathbf{x}_1, \ldots, \mathbf{x}_T$, the basic assumption in an SSM is that the joint probability of the hidden states and observations factors in the following way:

$$p(\mathbf{x}_{1:T}, \mathbf{y}_{1:T}|\theta) = \prod_{t=1}^{T} p(\mathbf{x}_t|\mathbf{x}_{t-1}, \theta)p(\mathbf{y}_t|\mathbf{x}_t, \theta) \tag{24}$$

In order words, the observations are assumed to have been generated from the hidden states via $p(\mathbf{y}_t|\mathbf{x}_t, \theta)$, and the hidden states are assumed to have first-order Markov dynamics captured by $p(\mathbf{x}_t|\mathbf{x}_{t-1}, \theta)$. We can consider the first term $p(\mathbf{x}_1|\mathbf{x}_0, \theta)$ to be a prior on the initial state of the system $\mathbf{x}_1$.

The simplest kind of state-space model assumes that all variables are multivariate Gaussian distributed and all the relationships are linear. In such *linear-Gaussian state-space models*, we can write

$$\mathbf{y}_t = C\mathbf{x}_t + \mathbf{v}_t \tag{25}$$

$$\mathbf{x}_t = A\mathbf{x}_{t-1} + \mathbf{w}_t \tag{26}$$

where the matrices C and A define the linear relationships and $\mathbf{v}$ and $\mathbf{w}$ are zero-mean Gaussian noise vectors with covariance matrices R and Q respectively. If we assume that the prior on the initial state $p(\mathbf{x}_1)$ is also Gaussian, then all subsequent $\mathbf{x}$s and $\mathbf{y}$s are also Gaussian due the the fact that Gaussian densities are closed under linear transformations. This model can be generalized in many ways, for example by augmenting it to include a sequence of observed inputs $\mathbf{u}_1, \ldots, \mathbf{u}_T$ as well as the observed model outputs $\mathbf{y}_1, \ldots, \mathbf{y}_T$, but we will not discuss generalizations further.

By comparing equations (11) and (25) we see that linear-Gaussian SSMs can be thought of as a time-series generalization of factor analysis where the factors are assumed to have linear-Gaussian dynamics over time.

The parameters of this model are $\theta = (A, C, Q, R)$. To learn ML settings of these parameters one can make use of the EM algorithm [9]. The E step of the algorithm involves computing $q(\mathbf{x}_{1:T}) = p(\mathbf{x}_{1:T}|\mathbf{y}_{1:T}, \theta)$ which is the posterior over hidden state sequences. In fact, this whole posterior does not have to be computed or represented, all that is required are the marginals $q(\mathbf{x}_t)$ and pairwise marginals $q(\mathbf{x}_t, \mathbf{x}_{t+1})$. These can be computed via the *Kalman smoothing algorithm*, which is an efficient algorithm for inferring the distribution over the hidden states of a linear-Gaussian SSM. Since the model is linear, the M step of the algorithm requires solving a pair of weighted linear regression problems to re-estimate A and C, while Q and R are estimated from the residuals of those regressions. This is analogous to the M step of factor analysis, which also involves solving a linear regression problem.

4.2 Hidden Markov Models (HMMs)

Hidden Markov models are similar to state-space models in that the sequence of observations is assumed to have been generated from a sequence of underlying

hidden states. The key difference is that in HMMs the state is assumed to be *discrete* rather than a continuous random vector. Let s_t denote the hidden state of an HMM at time t. We assume that s_t can take discrete values in $\{1, \ldots, K\}$. The model can again be written as in (24):

$$P(s_{1:T}, \mathbf{y}_{1:T}|\theta) = \prod_{t=1}^{T} P(s_t|s_{t-1}, \theta)P(\mathbf{y}_t|s_t, \theta) \tag{27}$$

where $P(s_1|s_0, \theta)$ is simply some initial distribution over the K settings of the first hidden state; we can call this discrete distribution $\boldsymbol{\pi}$, represented by a $K \times 1$ vector. The state-transition probabilities $P(s_t|s_{t-1}, \theta)$ are captured by a $K \times K$ transition matrix A, with elements $A_{ij} = P(s_t = i|s_{t-1} = j, \theta)$. The observations in an HMM can be either continuous or discrete. For continuous observations $\mathbf{y}_t$ one can for example choose a Gaussian density; thus $p(\mathbf{y}_t|s_t = i, \theta)$ would be a different Gaussian for each choice of $i \in \{1, \ldots, K\}$. This model is the dynamical generalization of a mixture of Gaussians. The marginal probability at each point in time is exactly a mixture of K Gaussians—the difference is that which component generates data point $\mathbf{y}_t$ and which component generated $\mathbf{y}_{t-1}$ are not independent random variables, but certain combinations are more and less probable depending on the entries in A. For $\mathbf{y}_t$ a discrete observation, let us assume that it can take on values $\{1, \ldots, L\}$. In that case the output probabilities $P(\mathbf{y}_t|s_t, \theta)$ can be captured by an $L \times K$ emission matrix, E.

The model parameters for a discrete-observation HMM are $\theta = (\boldsymbol{\pi}, A, E)$. Maximum likelihood learning of the model parameters can be approached using the EM algorithm, which in the case of HMMs is known as the *Baum-Welch algorithm*. The E step involves computing $Q(s_t)$ and $Q(s_t, s_{t+1})$ which are marginals of $Q(s_{1:T}) = P(s_{1:T}|\mathbf{y}_{1:T}, \theta)$. These marginals are computed as part of the *forward–backward algorithm* which as the name suggests sweeps forward and backward through the time series, and applies Bayes rule efficiently using the Markov conditional independence properties of the HMM, to compute the required marginals. The M step of HMM learning involves re-estimating $\boldsymbol{\pi}$, A, and E by adding up and normalizing expected counts for transitions and emissions that were computed in the E step.

4.3 Modeling Other Structured Data

We have considered the case of i.i.d. data and time series data. The observations in real world data sets can have many other possible structures as well. Let us mention a few examples, although it is not possible to strive for completeness.

In spatial data, the points are assumed to live in some metric, often Euclidean, space. Three examples of spatial data include epidemiological data which can be modeled as a function of the spatial location of the measurement; data from computer vision where the observations are measurements of features on a 2D input to the camera; and functional neuroimaging where the data can be physiological measurements related to neural activity located in 3D voxels defining coordinates in the brain. Generalizing HMMs, one can define Markov random

field models where there are a set of hidden variables correlated to neighbors in some lattice, and related to the observed variables.

Hierarchical or tree-structured data contains known or unknown tree-like correlation structure between the data points or measured features. For example, the data points may be features of animals related through an evolutionary tree. A very different form of structured data is if each data point itself is tree-structured, for example if each point is a parse tree of a sentence in the English language.

Finally, one can take the structured dependencies between variables and consider the structure itself as an unknown part of the model. Such models are known as *probabilistic relational models* and are closely related to graphical models which we will discuss in Section 7.

5 Nonlinear, Factorial, and Hierarchical Models

The models we have described so far are attractive because they are relatively simple to understand and learn. However, their simplicity is also a limitation, since the intricacies of real-world data are unlikely to be well-captured by a simple statistical model. This motivates us to seek to describe and study learning in much more flexible models.

A simple combination of two of the ideas we have described for i.i.d. data is the *mixture of factor analyzers* [10, 11, 12]. This model performs simultaneous clustering and dimensionality reduction on the data, by assuming that the covariance in each Gaussian cluster can be modeled by an FA model. Thus, it becomes possible to apply a mixture model to very high dimensional data while allowing each cluster to span a different sub-space of the data.

As their name implies linear-Gaussian SSMs are limited by assumptions of linearity and Gaussian noise. In many realistic dynamical systems there are significant nonlinear effects, which make it necessary to consider learning in *nonlinear state-space models*. Such models can also be learned using the EM algorithm, but the E step must deal with inference in non-Gaussian and potentially very complicated densities (since non-linearities will turn Gaussians into non-Gaussians), and the M step is nonlinear regression, rather than linear regression [13]. There are many methods of dealing with inference in non-linear SSMs, including methods such as particle filtering [14, 15, 16, 17, 18, 19], linearization [20], the unscented filter [21, 22], the EP algorithm [23], and embedded HMMs [24].

Non-linear models are also important if we are to consider generalizing simple dimensionality reduction models such as PCA and FA. These models are limited in that they can only find a linear subspace of the data to capture the correlations between the observed variables. There are many interesting and important nonlinear dimensionality reduction models, including generative topographic mappings (GTM) [25] (a probabilistic alternative to Kohonen maps), multi-dimensional scaling (MDS) [26, 27], principal curves [28], Isomap [29], and locally linear embedding (LLE) [30].

Hidden Markov models also have their limitations. Even though they can model nonlinear dynamics by discretizing the hidden state space, an HMM with K hidden states can only capture $\log_2 K$ bits of information in its state variable about the past of the sequence. HMMs can be extended by allowing a *vector* of discrete state variables, in an architecture known as a *factorial HMM* [31]. Thus a vector of M variables, each of which can take K states, can capture K^M possible states in total, and $M \log_2 K$ bits of information about the past of the sequence. The problem is that such a model, if dealt with naively as an HMM would have exponentially many parameters and would take exponentially long to do inference in. Both the complexity in time and number of parameters can be alleviated by restricting the interactions between the hidden variables at one time step and at the next time step. A generalization of these ideas is the notion of a *dynamical Bayesian network (DBN)* [32].

A relatively old but still quite powerful class of models for binary data is the *Boltzmann machine* (BM) [33]. This is a simple model inspired from Ising models in statistical physics. A BM is a multivariate model for capturing correlations and higher order statistics in vectors of binary data. Consider data consisting of vectors of M binary variables (t he elements of the vector may, for example, be pixels in a black-and-white image). Clearly, each data point can be an instance of one of 2^M possible patterns. An arbitrary distribution over such patterns would require a table with $2^M - 1$ entries, again intractable in number of parameters, storage, and computation time. A BM allows one to define flexible distributions over the 2^M entries of this table by using $\mathcal{O}(M^2)$ parameters defining a symmetric matrix of weights connecting the variables. This can be augmented with hidden variables in order to enrich the model class, without adding exponentially many parameters. These hidden variables can be organized into layers of a hierarchy as in the Helmholtz machine [34]. Other hierarchical models include recent generalizations of ICA designed to capture higher order statistics in images [35].

6 Intractability

The problem with the models described in the previous section is that learning their parameters is in general computationally intractable. In a model with exponentially many settings for the hidden states, doing the E step of an EM algorithm would require computing appropriate marginals of a distribution over exponentially many possibilities.

Let us consider a simple example. Imagine we have a vector of N binary random variables $\mathbf{s} = (s_1, \ldots, s_N)$, where $s_i \in \{0, 1\}$ and a vector of N known integers $(r_1, \ldots, r_N)$ where $r_i \in \{1, 2, 3, \ldots, 10\}$. Let the variable $Y = \sum_{i=1}^{N} r_i s_i$. Assume that the binary variables are all independent and identically distributed with $P(s_i = 1) = 1/2, \forall i$. Let N be 100. Now imagine that we are told $Y = 430$. How do we compute $P(s_i = 1 | Y = 430)$? The problem is that even though the s_i were independent *before* we observed the value of Y, now that we know the value of Y, not all settings of $\mathbf{s}$ are possible anymore. To figure out for some s_i

the probability of $P(s_i = 1|Y = 430)$ requires that we enumerate all potentially exponentially many ways of achieving $Y = 430$ and counting how many of those had $s_i = 1$ vs $s_i = 0$.

This example illustrates the following ideas: Even if the prior is simple, the posterior can be very complicated. Whether two random variables are independent or not is a function of one's state of knowledge. Thus s_i and s_j may be independent if we are not told the value of Y but are certainly dependent given the value of Y. These type of phenomena are related to "explaining-away" which refers to the fact that if there are multiple potential causes for some effect, observing one, explains away the need for the others [36].

Intractability can thus occur if we have a model with discrete hidden variables which can take on exponentially many combinations. Intractability can also occur with continuous hidden variables if their density is not simply described, or if they interact with discrete hidden variables. Moreover, even for simple models, such as a mixture of Gaussians, intractability occurs when we consider the parameters to be unknown as well, and we attempt to do Bayesian inference on them. To deal with intractability it is essential to have good tools for representing multivariate distributions, such as graphical models.

7 Graphical Models

Graphical models are an important tool for representing the dependencies between random variables in a probabilistic model. They are important for two reasons. First, graphs are an intuitive way of visualizing dependencies. We are used to graphical depictions of dependency, for example in circuit diagrams and in phylogenetic trees. Second, by exploiting the structure of the graph it is possible to devise efficient message passing algorithms for computing marginal and conditional probabilities in a complicated model. We discuss message passing algorithms for inference in Section 8.

The main statistical property represented explicitly by the graph is *conditional independence* between variables. We say that X and Y are conditionally independent given Z, if $P(X, Y|Z) = P(X|Z)P(Y|Z)$ for all values of the variables X,Y, and Z where these quantities are defined (i.e. excepting settings z where $P(Z = z) = 0$). We use the notation $X \perp\!\!\!\perp Y|Z$ to denote the above conditional independence relation. Conditional independence generalists to sets of variables in the obvious way, and it is different from *marginal independence* which states that $P(X, Y) = P(X)P(Y)$, and is denoted $X \perp\!\!\!\perp Y$.

There are several different graphical formalisms for depicting conditional independence relationships. We focus on three of the main ones: undirected, factor, and directed graphs.

7.1 Undirected Graphs

In an *undirected graphical model* each random variable is represented by a node, and the edges of the graph indicate conditional independence relationships.

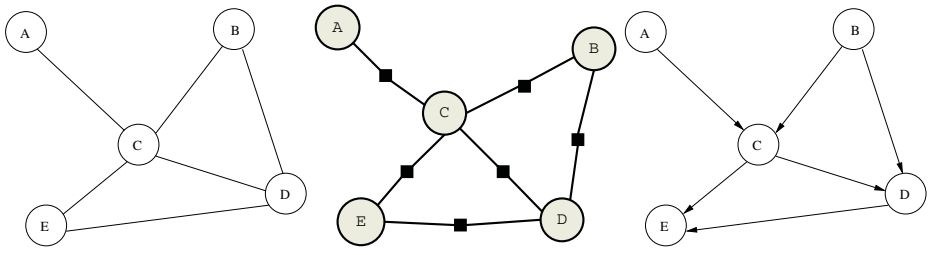

Fig. 1. Three kinds of probabilistic graphical model: undirected graphs, factor graphs and directed graphs

Specifically, let $\mathcal{X}, \mathcal{Y}$, and $\mathcal{Z}$ be sets of random variables. Then $\mathcal{X} \perp\!\!\!\perp \mathcal{Y} | \mathcal{Z}$ if every path on the graph from a node in $\mathcal{X}$ to a node in $\mathcal{Y}$ has to go through a node in $\mathcal{Z}$. Thus a variable X is conditionally independent of all other variables given the neighbors of X, and we say that the neighbors *separate* X from the rest of the graph. An example of an undirected graph is shown in Figure 1. In this graph $A \perp\!\!\!\perp B | C$ and $B \perp\!\!\!\perp E | \{C, D\}$, for example, and the neighbors of D are B, C, E.

A *clique* is a fully connected subgraph of a graph. A *maximal clique* is not contained in any other clique of the graph. It turns out that the set of conditional independence relations implied by the separation properties in the graph are satisfied by probability distributions which can be written as a normalized product of non-negative functions over the variables in the maximal cliques of the graph (this is known as the Hammersley-Clifford Theorem [37]). In the example in Figure 1, this implies that the probability distribution over (A, B, C, D, E) can be written as:

$$P(A, B, C, D, E) = c\, g_1(A, C) g_2(B, C, D) g_3(C, D, E) \tag{28}$$

Here, c is the constant that ensures that the probability distribution sums to 1, and g_1, g_2 and g_3 are non-negative functions of their arguments. For example, if all the variables are binary the function g_2 is a table with a non-negative number for each of the $8 = 2 \times 2 \times 2$ possible settings of the variables B, C, D. These non-negative functions are supposed to represent how compatible these settings are with each other, with a 0 encoding logical incompatibility. For this reason, the g's are sometimes referred to as *compatibility functions*, other times as *potential functions*. Undirected graphical models are also sometimes referred to as *Markov networks*.

7.2 Factor Graphs

In a *factor graph* there are two kinds of nodes, *variable nodes* and *factor nodes*, usually denoted as open circles and filled dots (Figure 1). Like an undirected model, the factor graph represents a factorization of the joint probability distribution: each factor is a non-negative function of the variables connected to the corresponding factor node. Thus for the factor graph in Figure 1 we have:

$$P(A,B,C,D,E) = cg_1(A,C)g_2(B,C)g_3(B,D),g_4(C,D)g_5(C,E)g_6(D,E) \tag{29}$$

Factor nodes are also sometimes called function nodes. Again, as in an undirected graphical model, the variables in a set $\mathcal{X}$ are conditionally independent of the variables in a set $\mathcal{Y}$ given $\mathcal{Z}$ if all paths from $\mathcal{X}$ to $\mathcal{Y}$ go through variables in $\mathcal{Z}$. Note that the factor graph is Figure 1 has exactly the same conditional independence relations as the undirected graph, even though the factors in the former are contained in the factors in the latter. Factor graphs are particularly elegant and simple when it comes to implementing message passing algorithms for inference (Section 8).

7.3 Directed Graphs

In *directed graphical models*, also known as probabilistic directed acyclic graphs (DAGs), belief networks, and Bayesian networks, the nodes represent random variables and the directed edges represent statistical dependencies. If there exists an edge from A to B we say that A is a *parent* of B, and conversely B is a *child* of A. A directed graph corresponds to the factorization of the joint probability into a product of the conditional probabilities of each node given its parents. For the example in Figure 1 we write:

$$P(A,B,C,D,E) = P(A)P(B)P(C|A,B)P(D|B,C)P(E|C,D) \tag{30}$$

In general we would write:

$$P(X_1,\ldots,X_N) = \prod_{i=1}^{N} P(X_i|X_{\mathrm{pa}_i}) \tag{31}$$

where X_{pa_i} denotes the variables that are parents of X_i in the graph.

Assessing the conditional independence relations in a directed graph is slightly less trivial than in undirected and factor graphs. Rather than simply looking at separation between sets of variables, one has to consider the directions of the edges. The graphical test for two sets of variables being conditionally independent given a third is called *d-separation* [36]. D-separation takes into account the following fact about *v-structures* of the graph, which consist of two (or more) parents of a child, as in the $A \rightarrow C \leftarrow B$ subgraph in Figure 1. In such a v-structure $A \perp\!\!\!\perp B$, but it is not true that $A \perp\!\!\!\perp B|C$. That is, A and B are marginally independent, but conditionally *dependent* given C. This can be easily checked by writing out $P(A,B,C) = P(A)P(B)P(C|A,B)$. Summing out C leads to $P(A,B) = P(A)P(B)$. However, given the value of C, $P(A,B|C) = P(A)P(B)P(C|A,B)/P(C)$ which does not factor into separate functions of A and B. As a consequence of this property of v-structures, in a directed graph a variable X is independent of all other variables given the parents of X, the children of X, and the parents of the children of X. This is the minimal set that d-separates X from the rest of the graph and is known as the Markov boundary for X.

It is possible, though not always appropriate, to interpret a directed graphical model as a causal generative model of the data. The following procedure would generate data from the probability distribution defined by a directed graph: draw a random value from the marginal distribution of all variables which do not have any parents (e.g. $a \sim P(A)$, $b \sim P(B)$), then sample from the conditional distribution of the children of these variables (e.g. $c \sim P(C|A = a, B = a)$), and continue this procedure until all variables are assigned values. In the model, $P(C|A, B)$ can capture the causal relationship between the causes A and B and the effect C. Such causal interpretations are much less natural for undirected and factor graphs, since even generating a sample from such models cannot easily be done in a hierarchical manner starting from "parents" to "children" except in special cases. Moreover, the potential functions capture mutual compatibilities, rather than cause-effect relations.

A useful property of directed graphical models is that there is no global normalization constant c. This global constant can be computationally intractable to compute in undirected and factor graphs. In directed graphs, each term is a conditional probability and is therefore already normalized $\sum_x P(X_i = x|X_{\mathrm{pa}_i}) = 1$.

7.4 Expressive Power

Directed, undirected and factor graphs are complementary in their ability to express conditional independence relationships. Consider the directed graph consisting of a single v-structure $A \rightarrow C \leftarrow B$. This graph encodes $A \perp\!\!\!\perp B$ but not $A \perp\!\!\!\perp B|C$. There exists no undirected graph or factor graph over these three variables which captures exactly these independencies. For example, in $A - C - B$ it is not true that $A \perp\!\!\!\perp B$ but it is true that $A \perp\!\!\!\perp B|C$. Conversely, if we consider the undirected graph in Figure 2, we see that some independence relationships are better captured by undirected models (and factor graphs).

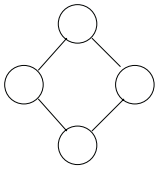

Fig. 2. No directed graph over 4 variables can represent the set of conditional independence relationships represented by this undirected graph

8 Exact Inference in Graphs

Probabilistic *inference* in a graph usually refers to the problem of computing the conditional probability of some variable X_i given the observed values of some other variables $X_{\mathrm{obs}} = x_{\mathrm{obs}}$ while marginalizing out all other variables. Starting from a joint distribution $P(X_1, \ldots, X_N)$, we can divide the set of all variables into three exhaustive and mutually exclusive sets $\{X_1, \ldots X_N\} = \{X_i\} \cup X_{\mathrm{obs}} \cup X_{\mathrm{other}}$. We wish to compute

$$P(X_i|X_{\text{obs}} = x_{\text{obs}}) = \frac{\sum_x P(X_i, X_{\text{other}} = x, X_{\text{obs}} = x_{\text{obs}})}{\sum_{x'} \sum_x P(X_i = x', X_{\text{other}} = x, X_{\text{obs}} = x_{\text{obs}})} \quad (32)$$

The problem is that the sum over x is exponential in the number of variables in X_{other}. For example. if there are M variables in X_{other} and each is binary, then there are 2^M possible values for x. If the variables are continuous, then the desired conditional probability is the ratio of two high-dimensional integrals, which could be intractable to compute. Probabilistic inference is essentially a problem of computing large sums and integrals.

There are several algorithms for computing these sums and integrals which exploit the structure of the graph to get the solution efficiently for certain graph structures (namely trees and related graphs). For general graphs the problem is fundamentally hard [38].

8.1 Elimination

The simplest algorithm conceptually is *variable elimination*. It is easiest to explain with an example. Consider computing $P(A = a|D = d)$ in the directed graph in Figure 1. This can be written

$$\begin{aligned}
P(A = a|D = d) &\propto \sum_c \sum_b \sum_e P(A = a, B = b, C = c, D = d, E = e) \\
&= \sum_c \sum_b \sum_e P(A = a)P(B = b)P(C = c|A = a, B = b) \\
&\qquad P(D = d|C = c, B = b)P(E = e|C = c, D = d) \\
&= \sum_c \sum_b P(A = a)P(B = b)P(C = c|A = a, B = b) \\
&\qquad P(D = d|C = c, B = b) \sum_e P(E = e|C = c, D = d) \\
&= \sum_c \sum_b P(A = a)P(B = b)P(C = c|A = a, B = b) \\
&\qquad P(D = d|C = c, B = b)
\end{aligned}$$

What we did was (1) exploit the factorization, (2) rearrange the sums, and (3) eliminate a variable, E. We could repeat this procedure and eliminate the variable C. When we do this we will need to compute a new function $\phi(A = a, B = b, D = d) \stackrel{\text{def}}{=} \sum_c P(C = c|A = a, B = b)P(D = d|C = c, B = b)$, resulting in:

$$P(A = a|D = d) \propto \sum_b P(A = a)P(B = b)\phi(A = a, B = b, D = d)$$

Finally, we eliminate B by computing $\phi'(A = a, D = d) \stackrel{\text{def}}{=} \sum_b P(B = b)\phi(A = a, B = b, D = d)$ to get our final answer which can be written

$$P(A = a|D = d) \propto P(A = a)\phi'(A = a, D = d) = \frac{P(A = a)\phi'(A = a, D = d)}{\sum_a P(A = a)\phi'(A = a, D = d)}$$

The functions we get when we eliminate variables can be thought of as messages sent by that variable to its neighbors. Eliminating transforms the graph by removing the eliminated node and drawing (undirected) edges between all the nodes in the Markov boundary of the eliminated node.

The same answer is obtained no matter what order we eliminate variables in; however, the computational complexity can depend dramatically on the ordering used.

8.2 Belief Propagation

The belief propagation (BP) algorithm is a message passing algorithm for computing conditional probabilities of any variable given the values of some set of other variables in a *singly-connected* directed acyclic graph [36]. The algorithm itself follows from the rules of probability and the conditional independence properties of the graph. Whereas variable elimination focuses on finding the conditional probability of a single variable X_i given $X_{\text{obs}} = x_{\text{obs}}$, belief propagation can compute at once all the conditionals $p(X_i|X_{\text{obs}} = x_{\text{obs}})$ for all i not observed.

We first need to define singly-connected directed graphs. A directed graph is singly connected if between every pair of nodes there is only one undirected path. An *undirected path* is a path along the edges of the graph ignoring the direction of the edges: in other words the path can traverse edges both upstream and downstream. If there is more than one undirected path between any pair of nodes then the graph is said to be *multiply connected*, or *loopy* (since it has loops).

Singly connected graphs have an important property which BP exploits. Let us call the set of observed variables the *evidence*, $e = X_{\text{obs}}$. Every node in the graph divides the evidence into upstream e^+_X and downstream e^-_X parts. For example, in Figure 3 the variables $U_1 \ldots U_n$ their parents, ancestors, and children and descendents (not including X, its children and descendents) and anything else connected to X via an edge directed toward X are all considered to be *upstream* of X; anything connected to X via an edge away from X is considered *downstream* of X (e.g. Y_1, its children, the parents of its children, etc). Similarly, every edge $X \rightarrow Y$ in a singly connected graph divides the

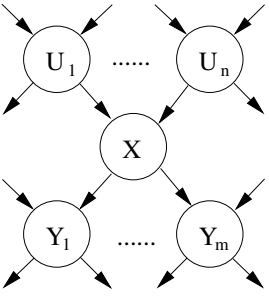

Fig. 3. Belief propagation in a directed graph

evidence into upstream and downstream parts. This separation of the evidence into upstream and downstream components does not generally occur in multiply-connected graphs.

Belief propagation uses three key ideas to compute the probability of some variable given the evidence $p(X|e)$, which we can call the "belief" about X.[4] First, the belief about X can be found by combining upstream and downstream evidence:

$$P(X|e) = \frac{P(X,e)}{P(e)} \propto P(X, e_X^+, e_X^-) \propto P(X|e_X^+)P(e_X^-|X) \tag{33}$$

The last proportionality results from the fact that given X the downstream and upstream evidence are conditionally independent: $P(e_X^-|X, e_X^+) = P(e_X^-|X)$. Second, the effect of the upstream and downstream evidence on X can be computed via a local message passing algorithm between the nodes in the graph. Third, the message from X to Y has to be constructed carefully so that node X doesn't send back to Y any information that Y sent to X, otherwise the message passing algorithm would reverberate information between nodes amplifying and distorting the final beliefs.

Using these ideas and the basic rules of probability we can arrive at the following equations, where $\mathrm{ch}(X)$ and $\mathrm{pa}(X)$ are children and parents of X, respectively:

$$\lambda(X) \stackrel{\text{def}}{=} P(e_X^-|X) = \prod_{j \in \mathrm{ch}(X)} P(e_{XY_j}^-|X) \tag{34}$$

$$\pi(X) \stackrel{\text{def}}{=} P(X|e_X^+) = \sum_{U_1 \ldots U_n} P(X|U_1, \ldots, U_n) \prod_{i \in \mathrm{pa}(X)} P(U_i|e_{U_i X}^+) \tag{35}$$

Finally, the messages from parents to children (e.g. X to Y_j) and the messages from children to parents (e.g. X to U_i) can be computed as follows:

$$\begin{aligned} \pi_{Y_j}(X) &\stackrel{\text{def}}{=} P(X|e_{XY_j}^+) \\ &\propto \Big[\prod_{k \neq j} P(e_{XY_k}^-|X)\Big] \sum_{U_1,\ldots,U_n} P(X|U_1 \ldots U_n) \prod_i P(U_i|e_{U_i X}^+) \end{aligned} \tag{36}$$

$$\begin{aligned} \lambda_X(U_i) &\stackrel{\text{def}}{=} P(e_{U_i X}^-|U_i) \\ &= \sum_X P(e_X^-|X) \sum_{U_k : k \neq i} P(X|U_1 \ldots U_n) \prod_{k \neq i} P(U_k|e_{U_k X}^+) \end{aligned} \tag{37}$$

It is important to notice that in the computation of both the top-down message (36) and the bottom-up message (37) the recipient of the message is explicitly excluded. Pearl's [36] mnemonic of calling these messages λ and π messages is meant to reflect their role in computing "likelihood" and "prior" terms.

[4] There is considerably variety in the field regarding the naming of algorithms. Belief propagation is also known as the sum-product algorithm, a name which some people prefer since beliefs seem subjective.

BP includes as special cases two important algorithms: Kalman smoothing for linear-Gaussian state-space models, and the forward–backward algorithm for hidden Markov models. Although BP is only valid on singly connected graphs there is a large body of research on its application to multiply connected graphs—the use of BP on such graphs is called *loopy belief propagation* and has been analyzed by several researchers [39, 40]. Interest in loopy belief propagation arose out of its impressive performance in decoding error correcting codes [41, 42, 43, 44]. Although the beliefs are not guaranteed to be correct on loopy graphs, interesting connections can be made to approximate inference procedures inspired by statistical physics known as the Bethe and Kikuchi free energies [45].

8.3 Factor Graph Propagation

In belief propagation, there is an asymmetry between the messages a child sends its parents and the messages a parent sends its children. Propagation in singly-connected factor graphs is conceptually much simpler and easier to implement. In a factor graph, the joint probability distribution is written as a product of factors. Consider a vector of variables $\mathbf{x} = (x_1, \ldots, x_n)$

$$p(\mathbf{x}) = p(x_1, \ldots, x_n) = \frac{1}{Z} \prod_j f_j(\mathbf{x}_{S_j}) \tag{38}$$

where Z is the normalisation constant, S_j denotes the subset of $\{1, \ldots, n\}$ which participate in factor f_j and $\mathbf{x}_{S_j} = \{x_i : i \in S_j\}$.

Let $\mathrm{n}(x)$ denote the set of factor nodes that are neighbours of x and let $\mathrm{n}(f)$ denote the set of variable nodes that are neighbours of f. We can compute probabilities in a factor graph by propagating messages from variable nodes to factor nodes and vice-versa. The message from variable x to function f is:

$$\mu_{x \to f}(x) = \prod_{h \in \mathrm{n}(x) \setminus \{f\}} \mu_{h \to x}(x) \tag{39}$$

while the message from function f to variable x is:

$$\mu_{f \to x}(x) = \sum_{\mathbf{x} \setminus x} \left(f(\mathbf{x}) \prod_{y \in \mathrm{n}(f) \setminus \{x\}} \mu_{y \to f}(y) \right) \tag{40}$$

Once a variable has received all messages from its neighbouring factor nodes we can compute the probability of that variable by multiplying all the messages and renormalising:

$$p(x) \propto \prod_{h \in \mathrm{n}(x)} \mu_{h \to x}(x) \tag{41}$$

Again, these equations can be derived by using Bayes rule and the conditional independence relations in a singly-connected factor graph. For multiply-connected factor graphs (where there is more than one path between at least one

pair of variable nodes) one can apply a loopy version of factor graph propagation. Since the algorithms for directed graphs and factor graphs are essentially based on the same ideas, we also call the loopy version of factor graph propagation "loopy belief propagation".

8.4 Junction Tree Algorithm

For multiply-connected graphs, the standard exact inference algorithms are based on the notion of a *junction tree* [46]. The basic idea of the junction tree algorithm is to group variables so as to convert the multiply-connected graph into a singly-connected undirected graph (tree) over sets of variables, and do inference in this tree.

We will not explain the algorithm in detail here, but rather give an overview of the steps involved. Starting from a directed graph, undirected edges are introduced between every pair of variables that share a child. This step is called "moralisation" in a tongue-in-cheek reference to the fact that it involves marrying the unmarried parents of every node. All the remaining edges are then changed from directed to undirected. We now have an undirected graph which does not imply any additional conditional or marginal independence relations which were not present in the original directed graph (although the undirected graph may easily have many fewer conditional or marginal independence relations than the directed graph). The next step of the algorithm is "triangulation" which introduces an edge cutting across every cycle of length 4. For example, the cycle $A - B - C - D - A$ which would look like Figure 2 would be triangulated either by adding an edge $A - C$ or an edge $B - D$. Once the graph has been triangulated, the maximal cliques of the graph are organised into a tree, where the nodes of the tree are cliques, by placing edges in the tree between some of the cliques with an overlap in variables (placing edges between all overlaps may not result in a tree). In general it may be possible to build several trees in this way, and triangulating the graph means than there exists a tree with the "running intersection property". This property ensures that none of the variable is represented in disjoint parts of the tree, as this would cause the algorithm to come up with multiple possibly inconsistent beliefs about the variable. Finally, once the tree with the running intersection property is built (the junction tree) it is possible to introduce the evidence into the tree and apply what is essentially a variant of belief propagation to this junction tree. This BP algorithm is operating on sets of variables contained in the cliques of the junction tree, rather than on individual variables in the original graph. As such, the complexity of the algorithm scales exponentially with the size of the largest clique in the junction tree. For example, if moralisation and triangulation results in a clique containing K binary variables, the junction tree algorithm would have to store and manipulate tables of size 2^K. Moreover, finding the optimal triangulation to get the most efficient junction tree for a particular graph is NP-complete [47, 48].

8.5 Cutest Conditioning

In certain graphs the simplest inference algorithm is *cutset conditioning* which is related to the idea of "reasoning by assumptions". The basic idea is very straightforward: find some small set of variables such that if they were given (i.e. you knew their values) it would make the remainder of the graph singly connected. For example, in the undirected graph in Figure 1, given C or D, the rest of the graph is singly connected. This set of variables is called the *cutset*. For each possible value of the variables in the cutset, run BP on the remainder of the graph to obtain the beliefs on the node of interest. These beliefs can be averaged with appropriate weights to obtain the true belief on the variable of interest. To make this more concrete, assume you want to find $P(X|e)$ and you discover a cutset consisting of a single variable C. Then

$$P(X|e) = \sum_c P(X|C = c, e)P(C = c\,|e) \tag{42}$$

where the beliefs $P(X|C = c, e)$ and corresponding weights $P(C = c\,|e)$ are computed as part of BP, run once for each value of c.

9 Learning in Graphical Models

In Section 8 we described exact algorithms for inferring the value of variables in a graph with known parameters and structure. If the parameters and structure are unknown they can be learned from the data [49]. The learning problem can be divided into learning the graph parameters for a known structure, and learning the model structure (i.e. which edges should be present or absent).[5]

We focus here on directed graphs with discrete variables, although some of these issues become much more subtle for undirected and factor graphs [50]. The parameters of a directed graph with discrete variables parameterise the conditional probability tables $P(X_i|X_{\mathrm{pa}_i})$. For each setting of X_{pa_i} this table contains a probability distribution over X_i. For example, if all variables are binary and X_i has K parents, then this *conditional probability table* has 2^{K+1} entries; however, since the probability over X_i has to sum to 1 for each setting of its parents there are only 2^K independent entries. The most general parameterisation would have a distinct parameter for each entry in this table, but this is often not a natural way to parameterise the dependency between variables. Alternatives (for binary data) are the noisy-or or sigmoid parameterisation of the dependencies [51]. Whatever the specific parameterisation, let $\boldsymbol{\theta}_i$ denote the parameters relating

[5] It should be noted that in Bayesian statistics there is no fundamental difference between parameters and variables, and therefore the learning and inference problems are really the same. All unknown quantities are treated as random variables, and learning is just inference about parameters and structure. It is however often useful to distinguish between parameters, which we assume to be fairly constant over the data, and variables, which we can assume to vary over each data point.

X_i to its parents, and let $\boldsymbol{\theta}$ denote all the parameters in the model. Let m denote the model structure, which corresponds to the set of edges in the graph. More generally the model structure can also contain the presence of additional hidden variables [52].

9.1 Learning Graph Parameters

We first consider the problem of learning graph parameters when the model structure is known and there are no missing or hidden variables. The presence of missing/hidden variables complicates the situation.

The Complete Data Case. Assume that the parameters controlling each *family* (a child and its parents) are distinct and that we observe N iid instances of all K variables in our graph. The data set is therefore $\mathcal{D} = \{X^{(1)} \dots X^{(N)}\}$ and the likelihood can be written

$$P(\mathcal{D}|\boldsymbol{\theta}) = \prod_{n=1}^{N} P(X^{(n)}|\boldsymbol{\theta}) = \prod_{n=1}^{N} \prod_{i=1}^{K} P(X_i^{(n)}|X_{\mathrm{pa}_i}^{(n)}, \boldsymbol{\theta}_i) \tag{43}$$

Clearly, maximising the log likelihood with respect to the parameters results in K decoupled optimisation problems, one for each family, since the log likelihood can be written as a sum of K independent terms. Similarly, if the prior factors over the $\boldsymbol{\theta}_i$, then the Bayesian posterior is also factored: $P(\boldsymbol{\theta}|\mathcal{D}) = \prod_i P(\boldsymbol{\theta}_i|\mathcal{D})$.

The Incomplete Data Case. When there is missing/hidden data, the likelihood no longer factors over the variables. Divide the variables in $X^{(n)}$ into observed and missing components, $X_{\mathrm{obs}}^{(n)}$ and $X_{\mathrm{mis}}^{(n)}$. The observed data is now $\mathcal{D} = \{X_{\mathrm{obs}}^{(1)} \dots X_{\mathrm{obs}}^{(N)}\}$ and the likelihood is:

$$P(\mathcal{D}|\boldsymbol{\theta}) = \prod_{n=1}^{N} P(X_{\mathrm{obs}}^{(n)}|\boldsymbol{\theta}) \tag{44}$$

$$= \prod_{n=1}^{N} \sum_{x_{\mathrm{mis}}^{(n)}} P(X_{\mathrm{mis}}^{(n)} = x_{\mathrm{mis}}^{(n)}, X_{\mathrm{obs}}^{(n)}|\boldsymbol{\theta}) \tag{45}$$

$$= \prod_{n=1}^{N} \sum_{x_{\mathrm{mis}}^{(n)}} \prod_{i=1}^{K} P(X_i^{(n)}|X_{\mathrm{pa}_i}^{(n)}, \boldsymbol{\theta}_i) \tag{46}$$

where in the last expression the missing variables are assumed to be set to the values $x_{\mathrm{mis}}^{(n)}$. Because of the missing data, the cost function can no longer be written as a sum of K independent terms and the parameters are all coupled. Similarly, even if the prior factors over the $\boldsymbol{\theta}_i$, the Bayesian posterior will couple all the $\boldsymbol{\theta}_i$.

One can still optimise the likelihood by making use of the EM algorithm (Section 3). The E step of EM infers the distribution over the hidden variables given

the current setting of the parameters. This can be done with BP for singly connected graphs or with the junction tree algorithm for multiply-connected graphs. In the M step, the objective function being optimised conveniently factors in exactly the same way as in the complete data case (c.f. Equation (21)). Whereas for the complete data case, the optimal ML parameters can often be computed in closed form, in the incomplete data case an iterative algorithm such as EM is usually required.

Bayesian parameter inference in the incomplete data case is also substantially more complicated. The parameters and missing data are coupled in the posterior distribution, as can be seen by multiplying (45) by the parameter prior and normalising. Inference can be achieved via approximate inference methods such as Markov chain Monte Carlo methods (Section 11.3, [53]) like Gibbs sampling, and variational approximations (Section 11.4, [54]).

9.2 Learning Graph Structure

There are two basic components to learning the structure of a graph from data: scoring and search. *Scoring* refers to computing a measure which can be used to compare different structures m and m' given a data set $\mathcal{D}$. *Search* refers to searching over the space of possible model structures, usually by proposing changes to the current model, so as to find the model with the highest score. This view of structure learning presupposes that the goal is to find a single structure with the highest score, although of course in the Bayesian inference framework it is desirable to infer the probability distribution over model structures given the data.

Scoring Metrics. Assume that you have a prior $P(m)$ over model structures, which is ideally based on some domain knowledge. The natural score to use is the probability of the model given the data (although see [55]) or some monotonic function of this:

$$s(m, \mathcal{D}) = P(m|\mathcal{D}) \propto P(\mathcal{D}|m)P(m). \tag{47}$$

This score requires computing the *marginal likelihood*

$$P(\mathcal{D}|m) = \int P(\mathcal{D}|\boldsymbol{\theta}, m)P(\boldsymbol{\theta}|m)d\boldsymbol{\theta}. \tag{48}$$

We discuss the intuitions behind the marginal likelihood as a natural score for model comparison in Section 10.

For directed graphical models with fully-observed discrete variables and factored Dirichlet priors over the parameters of the conditional probability tables, the integral in (48) is analytically tractable. For models with missing/hidden data, alternative choices of priors and types of variables, the integral in (48) is often intractable and approximation methods are required. Some of the standard approximations that can be applied in this context and many other Bayesian inference problems are briefly reviewed in Section 11.

Search Algorithms. Given a way of scoring models, one can search over the space of all possible valid graphical models for the one with the highest score [56]. The space of all possible graphs is very large (exponential in the number of variables) and for directed graphs it can be expensive to check whether a particular change to the graph will result in a cycle being formed. Thus intelligent heuristics are needed to search the space efficiently [57]. An alternative to trying to find the most probable graph are methods that sample over the posterior distribution of graphs [58]. This has the advantage that it avoids the problem of overfitting which can occur for algorithms that select a single structure with highest score out of exponentially many.

10 Bayesian Model Comparison and Occam's Razor

So far in this chapter we have seen many different kinds of models. One of the most important problems in unsupervised learning is automatically determining which models are appropriate for a given data set. Model selection and comparison questions include all of the following:

- Are there clusters in the data and if so, how many? What are their shapes (e.g. Gaussian, t-distributed)?
- Does the data live on a low dimensional manifold? What dimensionality? Is this manifold flat or curved?
- Is the data discretised? If so, to what precision?
- Is the data a time series? If so, is it better modelled by an HMM, a state-space model? Linear or nonlinear? Gaussian or non-Gaussian noise? How many states should the HMM have? How many state variables should the SSM have?
- Can the data be modelled well by a directed graph? What is the structure of this graph? Does it have hidden variables? Are these continuous or discrete?

Clearly, this list could go on. A human may be able to answer these questions via careful use of visualisation, hypothesis testing, and guesswork. But ultimately, an intelligent unsupervised learning system should be able to answer all these questions automatically.

Fortunately, the framework of Bayesian inference can be used to provide a rational, coherent and automatic way of answering all of the above questions. This means that, *given a complete specification of the prior assumptions* there is an automatic procedure (based on Bayes rule) which provides a unique answer. Of course, as always, if the prior assumptions are very poor, the answers obtained could be useless. Therefore, it is essential to think carefully about the prior assumptions before turning the automatic Bayesian handle.

Let us go over this automatic procedure. Consider a model m_i coming from a set of possible models $\{m_1, m_2, m_3, \ldots\}$. For instance, the model m_i might correspond to a Gaussian mixture model with i components. The models need not be nested, nor does the space of models need to be discrete (although we'll

focus on that case). Given data $\mathcal{D}$, the natural way to compare models is via their probability:

$$P(m_i|\mathcal{D}) = \frac{P(\mathcal{D}|m_i)P(m_i)}{P(\mathcal{D})} \tag{49}$$

To compare models, the denominator, which sums over the potentially huge space of all possible models, $P(\mathcal{D}) = \sum_j P(\mathcal{D}|m_j)P(m_j)$ is not required. Prior preference for models can be included in $P(m_i)$. However, it is interesting to look closely at the *marginal likelihood* term (sometimes called the *evidence* for model m_i). Assume that model m_i has parameters $\boldsymbol{\theta}_i$ (e.g. the means and covariance matrices of the i Gaussians, along with the mixing proportions, c.f. Section 2.4). The marginal likelihood integrates over all possible parameter values

$$P(\mathcal{D}|m_i) = \int P(\mathcal{D}|\boldsymbol{\theta}_i, m_i)P(\boldsymbol{\theta}|m_i)\, d\boldsymbol{\theta}_i \tag{50}$$

where $P(\boldsymbol{\theta}|m_i)$ is the prior over parameters, which is required for a complete specification of the model m_i.

The marginal likelihood has a very interesting interpretation. It is the probability of generating data set $\mathcal{D}$ from parameters that are *randomly sampled* from under the prior for m_i. This should be contrasted with the maximum likelihood for m_i which is the probability of the data under the single setting of the parameters $\hat{\boldsymbol{\theta}}_i$ that maximises $P(\mathcal{D}|\boldsymbol{\theta}_i, m_i)$. Clearly a more complicated model will have a higher maximum likelihood, which is the reason why maximising the likelihood results in *overfitting* — i.e. a preference for more complicated models than necessary. In contrast, the marginal likelihood can decrease as the model becomes more complicated. In a more complicated model sampling random parameter values can generate a wider range of possible data sets, but since the probability over data sets has to integrate to 1 (assuming a fixed number of data points) spreading the density to allow for more complicated data sets necessarily results in some simpler data sets having lower density under the model. This situation is diagrammed in Figure 4. The decrease in the marginal likelihood as additional parameters are added has been called the *automatic Occam's Razor* [59, 60, 61].

In theory all the questions posed at the beginning of this section could be addressed by defining appropriate priors and carefully computing marginal likelihoods of competing hypotheses. However, in practice the integral in (50) is usually very high dimensional and intractable. It is therefore necessary to approximate it.

11 Approximating Posteriors and Marginal Likelihoods

There are many ways of approximating the marginal likelihood of a model, and the corresponding parameter posterior. In this section, we review some of the most frequently used methods.

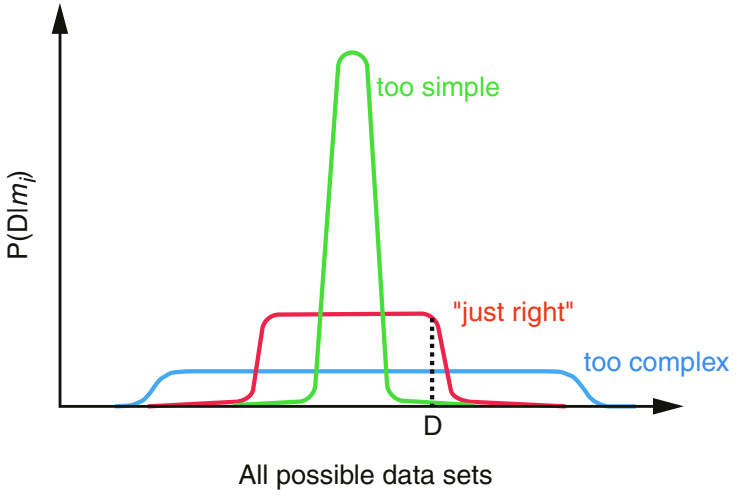

Fig. 4. The marginal likelihood (evidence) as a function of an abstract one dimensional representation of "all possible" data sets of some size N. Because the evidence is a probability over data sets, it must normalise to one. Therefore very complex models which can account for many datasets only achieve modest evidence; simple models can reach high evidences, but only for a limited set of data. When a dataset $\mathcal{D}$ is observed, the evidence can be used to select between model complexities

11.1 Laplace Approximation

It can be shown that under some regularity conditions, for large amounts of data N relative to the number of parameters in the model, d, the parameter posterior is approximately Gaussian around the MAP estimate, $\hat{\theta}$:

$$p(\boldsymbol{\theta}|\mathcal{D}, m) \approx (2\pi)^{-\frac{d}{2}}|A|^{\frac{1}{2}} \exp\left\{-\frac{1}{2}(\boldsymbol{\theta}-\hat{\boldsymbol{\theta}})^{\top} A\,(\boldsymbol{\theta}-\hat{\boldsymbol{\theta}})\right\} \tag{51}$$

Here A is the $d \times d$ negative of the Hessian matrix which measures the curvature of the log posterior at the MAP estimate:

$$A_{ij} = -\left.\frac{d^2}{d\theta_i d\theta_j} \log p(\boldsymbol{\theta}|\mathcal{D}, m)\right|_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}} \tag{52}$$

The matrix A is also referred to as the *observed information matrix*. Equation (51) is the *Laplace approximation* to the parameter posterior.

By Bayes rule, the marginal likelihood satisfies the following equality at any $\boldsymbol{\theta}$:

$$p(\mathcal{D}|m) = \frac{p(\boldsymbol{\theta}, \mathcal{D}|m)}{p(\boldsymbol{\theta}|\mathcal{D}, m)} \tag{53}$$

The Laplace approximation to the marginal likelihood can be derived by evaluating the log of this expression at $\hat{\boldsymbol{\theta}}$, using the Gaussian approximation to the posterior from equation (51) in the denominator:

$$\log p(\mathcal{D}|m) \approx \log p(\hat{\boldsymbol{\theta}}|m) + \log p(\mathcal{D}|\hat{\boldsymbol{\theta}}, m) + \frac{d}{2}\log 2\pi - \frac{1}{2}\log|A| \tag{54}$$

11.2 The Bayesian Information Criterion (BIC)

One of the disadvantages of the Laplace approximation is that it requires computing the determinant of the Hessian matrix. For models with many parameters, the Hessian matrix can be very large, and computing its determinant can be prohibitive.

The Bayesian Information Criterion (BIC) is a quick and easy way to compute an approximation to the marginal likelihood. BIC can be derived from the Laplace approximation by dropping all terms that do not depend on N, the number of data points. Starting from equation (54), we note that the first and third terms are constant with respect to the number of data points. Referring to the definition of the Hessian, we can see that its elements grow linearly with N. In the limit of large N we can therefore write $A = N\tilde{A}$, where $\tilde{A}$ is a matrix independent of N. We use the fact that for any scalar c and $d \times d$ matrix P, the determinant $|cP| = c^d|P|$, to get

$$\frac{1}{2}\log|A| \approx \frac{d}{2}\log N + \frac{1}{2}\log|\tilde{A}| \tag{55}$$

The last term does not grow with N, so by dropping it and substituting into Eq. (54) we get the BIC approximation:

$$\log p(\mathcal{D}|m) \approx \log p(\mathcal{D}|\hat{\boldsymbol{\theta}}, m) - \frac{d}{2}\log N \tag{56}$$

This expression is extremely easy to compute. Since the expression does not involve the prior it can be used either when $\hat{\boldsymbol{\theta}}$ is the MAP or the ML parameter estimate, the latter choice making the entire procedure independent of a prior. The likelihood is penalised by a term that depends linearly on the number of parameters in the model; this term is referred to as the *BIC penalty*. This is how BIC approximates the Bayesian Occam's Razor effect which penalises overcomplex models. The BIC criterion can also be derived from within the *Minimum Description Length* (MDL) framework.

The BIC penalty is clearly attractive since it does not require any costly integrals or matrix inversions. However this simplicity comes at a cost in accuracy which can sometimes be catastrophic. One of the dangers of BIC is that it relies on the number of parameters. The basic assumption underlying BIC, that the Hessian converges to N times a full-rank matrix, only holds for models in which all parameters are identifiable and well-determined. This is often not true.

11.3 Markov Chain Monte Carlo (MCMC)

Monte Carlo methods are a standard and often extremely effective way of computing complicated high dimensional integrals and sums. Many Bayesian inference problems can be seen as computing the integral (or sum) of some function $f(\boldsymbol{\theta})$ under some probability density $p(\boldsymbol{\theta})$:

$$\bar{f} \stackrel{\text{def}}{=} \int f(\boldsymbol{\theta}) p(\boldsymbol{\theta})\, d\boldsymbol{\theta}. \tag{57}$$

For example, the marginal likelihood is the integral of the likelihood function under the prior. Simple Monte Carlo approximates (57) by sampling M independent draws $\boldsymbol{\theta}_i \sim p(\boldsymbol{\theta})$ and computing the sample average of f:

$$\bar{f} \approx \frac{1}{M} \sum_{i=1}^{M} f(\boldsymbol{\theta}_i) \tag{58}$$

There are many limitations of simple Monte Carlo, for example it is often not possible to draw directly from p. Generalisations of simple Monte Carlo such as rejection sampling and importance sampling attempt to overcome some of these limitations.

An important family of generalisations of Monte Carlo methods are Markov chain Monte Carlo (MCMC) methods. These are commonly used and powerful methods for approximating the posterior over parameters and the marginal likelihood. Unlike simple Monte Carlo methods, the samples are not drawn independently but rather *dependently* in the form of a Markov chain $\ldots \boldsymbol{\theta}_i \rightarrow \boldsymbol{\theta}_{i+1} \rightarrow \boldsymbol{\theta}_{t+2} \ldots$ where each sample depends on the value of the previous sample. MCMC estimates have the property that the asymptotic distribution of $\boldsymbol{\theta}_i$ is the desired distribution. That is, $\lim_{t\rightarrow\infty} p_t(\boldsymbol{\theta}_t) = p(\boldsymbol{\theta})$. Creating MCMC methods is somewhat of an art, and there are many MCMC methods available, some of which are reviewed in [53]. Some notable examples are Gibbs sampling, the Metropolis algorithm, and Hybrid Monte Carlo.

11.4 Variational Approximations

Variational methods can be used to derive a family of lower bounds on the marginal likelihood and to perform approximate Bayesian inference over the parameters of a probabilistic models [62, 63, 64]. Variational methods provide an alternative to the asymptotic and sampling-based approximations described above; they tend to be more accurate than the asymptotic approximations like BIC and faster than the MCMC approaches.

Let $\mathbf{y}$ denote the observed variables, $\mathbf{x}$ denote the latent variables, and $\boldsymbol{\theta}$ denote the parameters. The log marginal likelihood of data $\mathbf{y}$ can be lower bounded by introducing any distribution over both latent variables and parameters which has support where $p(\mathbf{x}, \boldsymbol{\theta}|\mathbf{y}, m)$ does, and then appealing to Jensen's inequality (due to the concavity of the logarithm function):

$$\ln p(\mathbf{y}|m) = \ln \int p(\mathbf{y}, \mathbf{x}, \boldsymbol{\theta}|m)\, d\mathbf{x}\, d\boldsymbol{\theta} = \ln \int q(\mathbf{x}, \boldsymbol{\theta}) \frac{p(\mathbf{y}, \mathbf{x}, \boldsymbol{\theta}|m)}{q(\mathbf{x}, \boldsymbol{\theta})}\, d\mathbf{x}\, d\boldsymbol{\theta} \tag{59}$$

$$\geq \int q(\mathbf{x}, \boldsymbol{\theta}) \ln \frac{p(\mathbf{y}, \mathbf{x}, \boldsymbol{\theta}|m)}{q(\mathbf{x}, \boldsymbol{\theta})}\, d\mathbf{x}\, d\boldsymbol{\theta}. \tag{60}$$

Maximising this lower bound with respect to the free distribution $q(\mathbf{x}, \boldsymbol{\theta})$ results in $q(\mathbf{x}, \boldsymbol{\theta}) = p(\mathbf{x}, \boldsymbol{\theta}|\mathbf{y}, m)$ which when substituted above turns the inequality into an equality (c.f. Section 3). This does not simplify the problem since evaluat-

ing the true posterior distribution $p(\mathbf{x}, \boldsymbol{\theta}|\mathbf{y}, m)$ requires knowing its normalising constant, the marginal likelihood. Instead we use a simpler, factorised approximation $q(\mathbf{x}, \boldsymbol{\theta}) = q_{\mathbf{x}}(\mathbf{x})q_{\boldsymbol{\theta}}(\boldsymbol{\theta})$:

$$\ln p(\mathbf{y}|m) \;\geq\; \int q_{\mathbf{x}}(\mathbf{x})q_{\boldsymbol{\theta}}(\boldsymbol{\theta}) \ln \frac{p(\mathbf{y}, \mathbf{x}, \boldsymbol{\theta}|m)}{q_{\mathbf{x}}(\mathbf{x})q_{\boldsymbol{\theta}}(\boldsymbol{\theta})}\, d\mathbf{x}\, d\boldsymbol{\theta} \;\stackrel{\text{def}}{=}\; F_m(q_{\mathbf{x}}(\mathbf{x}), q_{\boldsymbol{\theta}}(\boldsymbol{\theta}), \mathbf{y}). \quad (61)$$

The quantity F_m is a functional of the free distributions, $q_{\mathbf{x}}(\mathbf{x})$ and $q_{\boldsymbol{\theta}}(\boldsymbol{\theta})$.

The variational Bayesian algorithm iteratively maximises F_m in equation (61) with respect to the free distributions, $q_{\mathbf{x}}(\mathbf{x})$ and $q_{\boldsymbol{\theta}}(\boldsymbol{\theta})$. We use elementary calculus of variations to take functional derivatives of the lower bound with respect to $q_{\mathbf{x}}(\mathbf{x})$ and $q_{\boldsymbol{\theta}}(\boldsymbol{\theta})$, each while holding the other fixed. This results in the following update equations where the superscript (t) denotes the iteration number:

$$q_{\mathbf{x}}^{(t+1)}(\mathbf{x}) \propto \exp\left[\int \ln p(\mathbf{x},\mathbf{y}|\boldsymbol{\theta}, m)\, q_{\boldsymbol{\theta}}^{(t)}(\boldsymbol{\theta})\, d\boldsymbol{\theta}\right] \quad (62)$$

$$q_{\boldsymbol{\theta}}^{(t+1)}(\boldsymbol{\theta}) \propto p(\boldsymbol{\theta}|m)\; \exp\left[\int \ln p(\mathbf{x},\mathbf{y}|\boldsymbol{\theta}, m)\, q_{\mathbf{x}}^{(t+1)}(\mathbf{x})\, d\mathbf{x}\right] \quad (63)$$

When there is more than one data point then there are different hidden variables $\mathbf{x}_i$ associated with each data point $\mathbf{y}_i$ and the step in (62) has to be carried out for each i, where the distributions are $q_{\mathbf{x}_i}^{(t)}(\mathbf{x}_i)$.

Clearly $q_{\boldsymbol{\theta}}(\boldsymbol{\theta})$ and $q_{\mathbf{x}_i}(\mathbf{x}_i)$ are coupled, so we iterate these equations until convergence. Recalling the EM algorithm (Section 3 and [65, 66]) we note the similarity between EM and the iterative algorithm in (62) and (63). This procedure is called the *Variational Bayesian EM Algorithm* and generalises the usual EM algorithm; see also [67] and [68].

Re-writing (61), it is easy to see that maximising F_m is equivalent to minimising the KL divergence between $q_{\mathbf{x}}(\mathbf{x})\; q_{\boldsymbol{\theta}}(\boldsymbol{\theta})$ and the joint posterior $p(\mathbf{x}, \boldsymbol{\theta}|\mathbf{y}, m)$:

$$\ln p(\mathbf{y}|m) - F_m(q_{\mathbf{x}}(\mathbf{x}), q_{\boldsymbol{\theta}}(\boldsymbol{\theta}), \mathbf{y}) = \int q_{\mathbf{x}}(\mathbf{x})\; q_{\boldsymbol{\theta}}(\boldsymbol{\theta}) \ln \frac{q_{\mathbf{x}}(\mathbf{x})\; q_{\boldsymbol{\theta}}(\boldsymbol{\theta})}{p(\boldsymbol{\theta}, \mathbf{x}|\mathbf{y}, m)}\, d\mathbf{x}\, d\boldsymbol{\theta} = \mathrm{KL}(q\|p) \quad (64)$$

Note that while this factorisation of the posterior distribution over latent variables and parameters may seem drastic, one can think of it as replacing stochastic dependencies between $\mathbf{x}$ and $\boldsymbol{\theta}$ with deterministic dependencies between relevant moments of the two sets of variables. To compare between models m and m' one can evaluate F_m and $F_{m'}$. This approach can, for example, be used to score graphical model structures [54].

Summarising, the variational Bayesian EM algorithm simultaneously computes an approximation to the marginal likelihood and to the parameter posterior by maximising a lower bound.

11.5 Expectation Propagation (EP)

Expectation propagation (EP; [23, 69]) is another powerful method for approximate Bayesian inference. Consider a Bayesian inference problem in which you

are given iid data $\mathcal{D} = \{\mathbf{x}^{(1)} \ldots, \mathbf{x}^{(N)}\}$ assumed to have come from a model $p(\mathbf{x}|\boldsymbol{\theta})$ parameterised by $\boldsymbol{\theta}$ with prior $p(\boldsymbol{\theta})$. The parameter posterior is:

$$p(\boldsymbol{\theta}|\mathcal{D}) = \frac{1}{p(\mathcal{D})} p(\boldsymbol{\theta}) \prod_{i=1}^{N} p(\mathbf{x}^{(i)}|\boldsymbol{\theta}). \tag{65}$$

To make the notation more general we can write the quantity we wish to approximate as a product of factors over $\boldsymbol{\theta}$,

$$\prod_{i=0}^{N} f_i(\boldsymbol{\theta}) = p(\boldsymbol{\theta}) \prod_{i=1}^{N} p(\mathbf{x}^{(i)}|\boldsymbol{\theta}) \tag{66}$$

where $f_0(\boldsymbol{\theta}) \stackrel{\text{def}}{=} p(\boldsymbol{\theta})$ and $f_i(\boldsymbol{\theta}) \stackrel{\text{def}}{=} p(\mathbf{x}^{(i)}|\boldsymbol{\theta})$ and we will ignore the normalising constants. We wish to approximate this by a product of *simpler* terms

$$q(\boldsymbol{\theta}) \stackrel{\text{def}}{=} \prod_{i=0}^{N} \tilde{f}_i(\boldsymbol{\theta}). \tag{67}$$

For example, consider a binary linear classification problem where $\boldsymbol{\theta}$ are the parameters of the classification hyperplane and $p(\boldsymbol{\theta})$ is a Gaussian prior ([69], Chapter 5). The true posterior is the product of this Gaussian and N likelihood terms, each of which defines a half-plane consistent with the class label observed. This posterior has a complicated shape, but we can approximate it using EP by assuming that each of the approximate likelihood terms $\tilde{f}_i$ is Gaussian in $\boldsymbol{\theta}$. Since the product of Gaussians is Gaussian, $q(\boldsymbol{\theta})$ will be a Gaussian approximation to the posterior. In general, one makes the approximate terms $\tilde{f}_i$ belong to some exponential family distribution so the overall approximation is in the same exponential family.

Having decided on the form of the approximation (67), let us consider how to tune this approximation so as to make it as accurate as possible. Ideally we would like to minimise the KL divergence between the true and the approximate distributions:

$$\min_{q(\boldsymbol{\theta})} \mathrm{KL}\left(\prod_{i=0}^{N} f_i(\boldsymbol{\theta}) \middle\| \prod_{i=0}^{N} \tilde{f}_i(\boldsymbol{\theta}) \right). \tag{68}$$

For example, if $q(\boldsymbol{\theta})$ is a Gaussian density, minimising this KL divergence will result in finding the *exact* mean and covariance of the true posterior distribution over parameters. Unfortunately, this KL divergence involves averaging with respect to the true posterior distribution, which will generally be intractable. Note that the KL divergence in Equation (68) is different from the KL minimised by variational Bayesian methods (64); the former averages with respect to the true distribution and is therefore usually intractable, while the latter averages with respect to the approximate distribution and is often tractable. Moreover, for exponential family approximations the former KL has a unique global optimum, while the latter usually has multiple local optima.

Since we cannot minimise (68) we can instead consider minimising the KL divergence between each true term and the corresponding approximate term. That is, for each i:

$$\min_{\tilde{f}_i(\boldsymbol{\theta})} \mathrm{KL}\left(f_i(\boldsymbol{\theta}) \| \tilde{f}_i(\boldsymbol{\theta})\right). \tag{69}$$

This will usually be much easier to do, but each such approximate term will result in some error. Multiplying all the approximate terms together will probably result in an unacceptably inaccurate approximation. On the plus side, this approach is non-iterative in that once each term is approximated they are simply multiplied to get a final answer.

The Expectation Propagation (EP) algorithm is an iterative procedure which is as easy as the naive approach in (69) but which results in a much more accurate approximation. At each step of EP, one of the terms is optimised in the context of all the other approximate terms, i.e. for each i:

$$\min_{\tilde{f}_i(\boldsymbol{\theta})} \mathrm{KL}\Big(f_i(\boldsymbol{\theta}) \prod_{j \neq i} \tilde{f}_j(\boldsymbol{\theta}) \Big\| \tilde{f}_i(\boldsymbol{\theta}) \prod_{j \neq i} \tilde{f}_j(\boldsymbol{\theta})\Big). \tag{70}$$

Since the approximate terms depend on each other, this procedure is iterated. On the left hand side of the KL divergence the i^{th} exact term is incorporated into $\prod_{j \neq i} \tilde{f}_j(\boldsymbol{\theta})$, which is assumed to be in the exponential family. The right hand side is an exponential-family approximation to this whole product. The minimisation is done by matching the appropriate moments (expectations) of $f_i(\boldsymbol{\theta}) \prod_{j \neq i} \tilde{f}_j(\boldsymbol{\theta})$. The name "Expectation Propagation" comes from the fact that each step corresponds to computing certain expectations, and the effect of these expectations is propagated to subsequent terms in the approximation. In fact, the messages in belief propagation can be derived as a particular form of EP where the approximating distribution is assumed to be a fully factored product of marginals over the variables in $\boldsymbol{\theta}$, i.e. $q(\boldsymbol{\theta}) = \prod_k q_k(\theta_k)$ [69].

In its simplest form, the EP algorithm can be summarised as in Figure 5. Although the algorithm as described here often converges, each step of the algorithm is not in fact decreasing any objective function so there is no guarantee of convergence. Convergent forms of EP can be derived by making use of the EP energy function [70] although these may not be as fast and simple to implement as the algorithm in Figure 5.

12 Conclusion

In this chapter, we have seen that unsupervised learning can be viewed from the perspective of statistical modelling. Statistics provides a coherent framework for learning from data and for reasoning under uncertainty. Many interesting statistical models used for unsupervised learning can be cast as latent variable models and graphical models. These types of models have played an important role in defining unsupervised learning systems for a variety of different kinds of data.

Input $f_0(\boldsymbol{\theta}) \dots f_N(\boldsymbol{\theta})$
Initialise $\tilde{f}_0(\boldsymbol{\theta}) = f_0(\boldsymbol{\theta})$, $\tilde{f}_i(\boldsymbol{\theta}) = 1$ for $i > 0$, $q(\boldsymbol{\theta}) = \prod_i \tilde{f}_i(\boldsymbol{\theta})$
repeat
  for $i = 0 \dots N$ **do**
    Deletion: $q_{\setminus i}(\boldsymbol{\theta}) \leftarrow \dfrac{q(\boldsymbol{\theta})}{\tilde{f}_i(\boldsymbol{\theta})} = \prod_{j \neq i} \tilde{f}_j(\boldsymbol{\theta})$
    Projection: $\tilde{f}_i^{\text{new}}(\boldsymbol{\theta}) \leftarrow \arg\min_{f(\boldsymbol{\theta})} \mathrm{KL}(f_i(\boldsymbol{\theta}) q_{\setminus i}(\boldsymbol{\theta}) \| f(\boldsymbol{\theta}) q_{\setminus i}(\boldsymbol{\theta}))$
    Inclusion: $q(\boldsymbol{\theta}) \leftarrow \tilde{f}_i^{\text{new}}(\boldsymbol{\theta})\, q_{\setminus i}(\boldsymbol{\theta})$
  end for
until convergence

Fig. 5. The EP algorithm. Some variations are possible: this assumes that f_0 is in the exponential family, and updates sequentially over i rather than randomly. The names for the steps (deletion, projection, inclusion) are not the same as in [69]

Graphical models have also played an important unifying framework for thinking about the role of conditional independence in inference in models with many variables. While for certain models exact inference is computationally tractable, for most of the models in this chapter we have seen that exact inference involves intractable sums and integrals. Thus, the study of unsupervised learning has lead us into focusing on ways of approximating high dimensional sums and integrals. We have reviewed many of the principal approximations, although of course in the limited space of this chapter one cannot hope to have a comprehensive review of approximation methods.

There are many interesting and relevant topics we did not get a chance to cover in this review of unsupervised learning. One of these is the interplay of unsupervised and supervised learning, in the form of semi-supervised learning. *Semi-supervised learning* refers to learning problems in which there is a small amount of labelled data and a large amount of unlabelled data. These problems are very natural, especially in domains where collecting data can be cheap (i.e. the internet) but labelling it can be expensive or time consuming. The key question in semi-supervised learning is how the data distribution from the unlabelled data should influence the supervised learning problem [71]. Many of the approaches to this problem attempt to infer a manifold, graph structure, or tree-structure from the unlabelled data and use spread in this structure to determine how labels will generalise to new unlabelled points [72, 73, 74, 75].

Another area of great interest which we did not have the space to cover are *nonparametric models*. The basic assumption of parametric statistical models is that the model is defined using a finite number of parameters. The number of parameters is assumed fixed regardless of the number of data points. Thus the parameters provide a finite summary of the data. In nonparametric models, the number of "parameters" in the model is allowed to grow with the size of the data set. With more data, the model becomes more complex, with no a-priori limit on the complexity of the model. For this reason nonparametric models

are also sometimes called *infinite models*. An important example of this are *infinite mixture models*, more formally known as *Dirichlet Process mixtures* [76, 77]. These correspond to mixture models (Section 2.4) where the number of components is assumed to be infinite. Inference can be done in these models using MCMC methods [78, 79, 80], variational methods [81], or the EP algorithm [82]. Just as hidden Markov models can be seen as an extension of finite mixture models to model time series data, it is possible to extend infinite mixture models to hidden Markov models with infinitely many states [83]. Infinite models based on Dirichlet processes have also been generalised to be hierarchical in several different ways [84, 85]. Bayesian inference in nonparametric models is one of the most active areas of research in unsupervised learning, and there still remain many open problems.

As we have seen, the field of unsupervised learning can be understood formally within the framework of information theory and statistics. However, it is important not to lose sight of the tremendous influence ideas from neuroscience and psychology have had on the field. Many of the models we have reviewed here started life as models of brain function. These models were inspired by the brain's ability to extract statistical patterns from sensory data and to recognise complex visual scenes, sounds, and odours. Unsupervised learning theory and algorithms still have a long way to go to mimic some of the learning abilities of biological brains. As the boundaries of unsupervised learning get pushed forward, we will hopefully not only benefit from better learning machines and also improve our understanding of how the brain learns.

References

1. MacKay, D.J.C.: Information Theory, Inference, and Learning Algorithms. Cambridge University Press (2003)
2. Jaynes, E.T.: Probability Theory: The Logic of Science (Edited by G. Larry Bretthorst). Cambridge University Press (2003)
3. Girosi, F., Jones, M., Poggio, T.: Regularization theory and neural networks architectures. Neural Computation **7** (1995) 219–269
4. Green, P.J.: Penalized likelihood. In: Encyclopedia of Statistical Sciences, Update Volume 2. (1998)
5. Roweis, S.T., Ghahramani, Z.: A unifying review of linear Gaussian models. Neural Computation **11** (1999) 305–345
6. Roweis, S.T.: EM algorithms for PCA and SPCA. In Jordan, M.I., Kearns, M.J., Solla, S.A., eds.: Advances in Neural Information Processing Systems. Volume 10., The MIT Press (1998)
7. Tipping, M.E., Bishop, C.M.: Probabilistic principal component analysis. Journal of the Royal Statistical Society, Series B **61** (1999) 611–622
8. Salakhutdinov, R., Roweis, S.T., Ghahramani, Z.: Optimization with EM and Expectation-Conjugate-Gradient. In: International Conference on Machine Learning (ICML-2003). (2003) 672–679
9. Shumway, R.H., Stoffer, D.S.: An approach to time series smoothing and forecasting using the EM algorithm. J. Time Series Analysis **3** (1982) 253–264

10. Ghahramani, Z., Hinton, G.E.: The EM algorithm for mixtures of factor analyzers. University of Toronto, Technical Report CRG-TR-96-1 (1996)
11. Hinton, G.E., Dayan, P., Revow, M.: Modeling the manifolds of images of handwritten digits. IEEE Trans. Neural Networks **8** (1997) 65–74
12. Tipping, M.E., Bishop, C.M.: Mixtures of probabilistic principal component analyzers. Neural Computation **11** (1999) 443–482
13. Ghahramani, Z., Roweis, S.T.: Learning nonlinear dynamical systems using an EM algorithm. In: NIPS 11. (1999) 431–437
14. Handschin, J.E., Mayne, D.Q.: Monte Carlo techniques to estimate the conditional expectation in multi-stage non-linear filtering. International Journal of Control **9** (1969) 547–559
15. Gordon, N.J., Salmond, D.J., Smith, A.F.M.: A novel approach to nonlinear/non-Gaussian Bayesian state space estimation. IEE Proceedings F: Radar and Signal Processing **140** (1993) 107–113
16. Kanazawa, K., Koller, D., Russell, S.J.: Stochastic simulation algorithms for dynamic probabilistic networks. In Besnard, P., Hanks, S., eds.: Uncertainty in Artificial Intelligence. Proceedings of the Eleventh Conference. Morgan Kaufmann Publishers, San Francisco, CA (1995) 346–351
17. Kitagawa, G.: Monte Carlo filter and smoother for non-Gaussian nonlinear state space models. J. of Computational and Graphical Statistics **5** (1996) 1–25
18. Isard, M., Blake, A.: Condensation – conditional density propagation for visual tracking (1998)
19. Doucet, A., de Freitas, J.F.G., Gordon, N.: Sequential Monte Carlo Methods in Practice. Springer-Verlag, New York (2000)
20. Anderson, B.D.O., Moore, J.B.: Optimal Filtering. Prentice-Hall, Englewood Cliffs, NJ (1979)
21. Julier, S.J., Uhlmann, J.K.: A new extension of the Kalman filter to nonlinear systems. In: Int. Symp. Aerospace/Defense Sensing, Simulation and Controls. (1997)
22. Wan, E.A., van der Merwe, R., Nelson, A.T.: Dual estimation and the unscented transformation. In: NIPS 12. (2000) 666–672
23. Minka, T.P.: Expectation propagation for approximate Bayesian inference. In: Uncertainty in Artificial Intelligence: Proceedings of the Seventeenth Conference (UAI-2001), San Francisco, CA, Morgan Kaufmann Publishers (2001) 362–369
24. Neal, R.M., Beal, M.J., Roweis, S.T.: Inferring state sequences for non-linear systems with embedded hidden Markov models. In Thrun, S., Saul, L., Schölkopf, B., eds.: Advances in Neural Information Processing Systems 16, Cambridge, MA, MIT Press (2004)
25. Bishop, C.M., Svensen, M., Williams, C.K.I.: GTM: The generative topographic mapping. Neural Computation **10** (1998) 215–234
26. Shepard, R.N.: The analysis of proximities: multidimensional scaling with an unknown distance function i and ii. Psychometrika **27** (1962) 125–139 and 219–246
27. Kruskal, J.B.: Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. Psychometrika **29** (1964) 1–27 and 115–129
28. Hastie, T., Stuetzle, W.: Principle curves. Journal of the American Statistical Association **84** (1989) 502–516
29. Tenenbaum, J.B., de Silva, V., Langford, J.C.: A global geometric framework for nonlinear dimensionality reduction. Science **290** (2000) 2319–2323
30. Roweis, S.T., Saul, L.K.: Nonlinear dimensionality reduction by locally linear embedding. Science **290** (2000) 2323–2326

31. Ghahramani, Z., Jordan, M.I.: Factorial hidden Markov models. Machine Learning **29** (1997) 245–273
32. Murphy, K.P.: Dynamic Bayesian Networks: Representation, Inference and Learning. PhD thesis, UC Berkeley, Computer Science Division (2002)
33. Ackley, D.H., Hinton, G.E., Sejnowski, T.J.: A learning algorithm for Boltzmann machines. Cognitive Science **9** (1985) 147–169
34. Hinton, G.E., Dayan, P., Frey, B.J., Neal, R.M.: The wake-sleep algorithm for unsupervised neural networks. Science **268** (1995) 1158–1161
35. Karklin, Y., Lewicki, M.S.: Learning higher-order structures in natural images. Network: Computation in Neural Systems **14** (2003) 483–499
36. Pearl, J.: Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufmann, San Mateo, CA (1988)
37. Besag, J.: Spatial interaction and the statistical analysis of lattice systems. Journal of the Royal Statistical Society. Ser. B **6** (1974) 192–236
38. Cooper, G.F.: The computational complexity of probabilistic inference using Bayesian belief networks. Artificial Intelligence **42** (1990) 393–405
39. Weiss, Y.: Correctness of local probability propagation in graphical models with loops. Neural Computation **12** (2000) 1–41
40. Weiss, Y., Freeman, W.T.: On the optimality 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** (2001)
41. Gallager, R.G.: Low-Density Parity-Check Codes. MIT Press, Cambridge, MA (1963)
42. Berrou, C., Glavieux, A., Thitimajshima, P.: Near shannon limit error-correcting coding and decoding: Turbo-codes (1). In: Proc. ICC '93. (1993) 1064–1070
43. McEliece, R.J., MacKay, D.J.C., Cheng, J.F.: Turbo decoding as an instance of Pearl's 'Belief Propagation' algorithm. IEEE Journal on Selected Areas in Communications **16** (1998) 140–152
44. MacKay, D.J.C., Neal, R.M.: Good error-correcting codes based on very sparse matrices. IEEE Transactions on Information Theory **45** (1999) 399–431
45. Yedidia, J.S., Freeman, W.T., Weiss, Y.: Generalized belief propagation. In: NIPS 13, Cambridge, MA, MIT Press (2001)
46. Lauritzen, S.L., Spiegelhalter, D.J.: Local computations with probabilities on graphical structures and their application to expert systems. J. Royal Statistical Society B (1988) 157–224
47. Arnborg, S., Corneil, D.G., Proskurowski, A.: Complexity of finding embeddings in a k-tree. SIAM Journal of Algebraic and Discrete Methods **8** (1987) 277–284
48. Kjaerulff, U.: Triangulation of graphs—algorithms giving small total state space (1990)
49. Heckerman, D.: A tutorial on learning with Bayesian networks. Technical Report MSR-TR-95-06, Microsoft Research (1996)
50. Murray, I., Ghahramani, Z.: Bayesian learning in undirected graphical models: Approximate MCMC algorithms. In: Proceedings of UAI. (2004)
51. Neal, R.M.: Connectionist learning of belief networks. Artificial Intelligence **56** (1992) 71–113
52. Elidan, G., Lotner, N., Friedman, N., Koller, D.: Discovering hidden variables: A structure-based approach. In: Advances in Neural Information Processing Systems (NIPS). (2001)
53. Neal, R.M.: Probabilistic inference using Markov chain Monte Carlo methods. Technical report, Department of Computer Science, University of Toronto (1993)

54. Beal, M.J., Ghahramani, Z.: The variational Bayesian EM algorithm for incomplete data: With application to scoring graphical model structures. In Bernardo, J.M., Dawid, A.P., Berger, J.O., West, M., Heckerman, D., Bayarri, M.J., eds.: Bayesian Statistics 7, Oxford University Press (2003)
55. Heckerman, D., Chickering, D.M.: A comparison of scientific and engineering criteria for Bayesian model selection (1996)
56. Friedman, N.: The Bayesian structural EM algorithm. In: Proc. Fourteenth Conference on Uncertainty in Artificial Intelligence (UAI '98), San Francisco, CA, Morgan Kaufmann (1998)
57. Moore, A., Wong, W.K.: Optimal reinsertion: A new search operator for accelerated and more accurate Bayesian network structure learning. In Fawcett, T., Mishra, N., eds.: Proceedings of the 20th International Conference on Machine Learning (ICML '03), Menlo Park, California, AAAI Press (2003) 552–559
58. Friedman, N., Koller, D.: Being Bayesian about network structure: A Bayesian approach to structure discovery in Bayesian networks. Machine Learning **50** (2003) 95–126
59. Jefferys, W., Berger, J.: Ockham's razor and Bayesian analysis. American Scientist **80** (1992) 64–72
60. MacKay, D.J.C.: Probable networks and plausible predictions—a review of practical Bayesian methods for supervised neural networks. Network: Computation in Neural Systems **6** (1995) 469–505
61. Rasmussen, C.E., Ghahramani, Z.: Occam's razor. In: Advances in Neural Information Processing Systems 13, Cambridge, MA, MIT Press (2001)
62. Jordan, M.I., Ghahramani, Z., Jaakkola, T.S., Saul, L.K.: An introduction to variational methods in graphical models. Machine Learning **37** (1999) 183–233
63. Winn, J.: Variational Message Passing and its Applications. PhD thesis, Department of Physics, University of Cambridge (2003)
64. Wainwright, M.J., Jordan, M.I.: Graphical models, exponential families, and variational inference. Technical Report 649, UC Berkeley, Dept. of Statistics (2003)
65. Dempster, A., Laird, N., Rubin, D.: Maximum likelihood from incomplete data via the EM algorithm. J. Royal Statistical Society Series B **39** (1977) 1–38
66. Neal, R.M., Hinton, G.E.: A new view of the EM algorithm that justifies incremental, sparse, and other variants. In Jordan, M.I., ed.: Learning in Graphical Models. Kluwer Academic Press (1998)
67. Attias, H.: Inferring parameters and structure of latent variable models by variational Bayes. In: Proc. 15th Conf. on Uncertainty in Artificial Intelligence. (1999)
68. Ghahramani, Z., Beal, M.J.: Propagation algorithms for variational Bayesian learning. In: Advances in Neural Information Processing Systems 13, Cambridge, MA, MIT Press (2001)
69. Minka, T.P.: A family of algorithms for approximate Bayesian inference. PhD thesis, MIT (2001)
70. Minka, T.P.: The EP energy function and minimization schemes. Technical report (2001)
71. Seeger, M.: Learning with labeled and unlabeled data. Technical report, University of Edinburgh (2001)
72. Szummer, M., Jaakkola, T.S.: Partially labeled classification with Markov random walks. In: NIPS. (2001)
73. Zhu, X., Ghahramani, Z., Lafferty, J.: Semi-supervised learning using Gaussian fields and harmonic functions. In: The Twentieth International Conference on Machine Learning (ICML-2003). (2003)

74. Belkin, M., Niyogi, P.: Semi-supervised learning on Riemannian manifolds. Machine Learning **56** (2004) 209–239
75. Kemp, C., Griffiths, T.L., Stromsten, S., Tenenbaum, J.B.: Semi-supervised learning with trees. In: NIPS 16. (2004)
76. Antoniak, C.E.: Mixtures of Dirichlet processes with applications to Bayesian nonparametric problems. Annals of Statistics **2** (1974) 1152–1174
77. Ferguson, T.S.: Bayesian density estimation by mixtures of normal distributions. In: Recent Advances in Statistics, New York, Academic Press (1983) 287–302
78. Escobar, M.D., West, M.: Bayesian density estimation and inference using mixtures. Journal of the American Statistical Association **90** (1995) 577–588
79. Neal, R.M.: Markov chain sampling methods for Dirichlet process mixture models. Journal of Computational and Graphical Statistics **9** (2000) 249–265
80. Rasmussen, C.E.: The infinite Gaussian mixture model. In: Adv. Neur. Inf. Proc. Sys. 12. (2000) 554–560
81. Blei, D., Jordan, M.I.: Variational methods for the Dirichlet process. In: Proceedings of the 21st International Conference on Machine Learning. (2004)
82. Minka, T.P., Ghahramani, Z.: Expectation propagation for infinite mixtures. Technical report, Presented at NIPS'03 Workshop on Nonparametric Bayesian Methods and Infinite Models (2003)
83. Beal, M., Ghahramani, Z., Rasmussen, C.: The infinite hidden Markov model. In: Advances in Neural Information Processing Systems. Volume 14., MIT Press (2001)
84. Neal, R.M.: Density modeling and clustering using Dirichlet diffusion trees. In et al., J.M.B., ed.: Bayesian Statistics 7. (2003) 619–629
85. Teh, Y.W., Jordan, M.I., Beal, M.J., Blei, D.M.: Hierarchical Dirichlet processes. Technical Report 653, Department of Statistics, University of California at Berkeley (2004)