Extending Expectation Propagation for Graphical Models

Yuan Qi

The MIT Media Laboratory
Cambridge, MA, 02139
October 10, 2004

Abstract

Graphical models have been widely used in many applications, ranging from human behavior recognition to wireless signal detection. However, efficient inference and learning techniques for graphical models are needed to handle complex models, such as hybrid Bayesian networks.

This thesis proposes extensions of expectation propagation, a powerful generalization of loopy belief propagation, to develop efficient Bayesian inference and learning algorithms for graphical models. The first two chapters of the thesis present inference algorithms for generative graphical models, and the next two propose learning algorithms for conditional graphical models.

First, the thesis proposes a window-based EP smoothing algorithm for online estimation on hybrid dynamic Bayesian networks. For an application in wireless communications, window-based EP smoothing achieves estimation accuracy comparable to sequential Monte Carlo methods, but with less than one-tenth computational cost.

Second, it develops a new method that combines tree-structured EP approximations with the junction tree for inference on loopy graphs. This new method saves computation and memory by propagating messages only locally to a subgraph when processing each edge in the entire graph. Using this local propagation scheme, this method is not only more accurate, but also faster than loopy belief propagation and structured variational methods.

Third, it proposes predictive automatic relevance determination (ARD) to enhance classification accuracy in the presence of irrelevant features. ARD is a Bayesian technique for feature selection. The thesis discusses the overfitting problem associated with ARD, and proposes a method that optimizes the estimated predictive performance, instead of maximizing the model evidence. For a gene expression classification problem, predictive ARD outperforms previous methods, including traditional ARD as well as support vector machines combined with feature selection techniques.

Finally, it presents Bayesian conditional random fields (BCRFs) for classifying interdependent and structured data, such as sequences, images or webs. BCRFs estimate the posterior distribution of model parameters and average prediction over this posterior to avoid overfitting. For the problems of frequently-asked-question labeling and of ink recognition, BCRFs achieve superior prediction accuracy over conditional random fields trained with maximum likelihood and maximum a posteriori criteria.

Thesis Supervisor: Rosalind W. Picard
Title: Associate Professor of Media Arts and Sciences

Doctoral Dissertation Committee

Extending Expectation Propagation for Graphical Models

Yuan Qi

Thesis Advisor .....

Rosalind W. Picard

Associate Professor of Media Arts and Sciences

Massachusetts Institute of Technology

Thesis Reader .....

Thomas P. Minka

Research Scientist of Microsoft Research

Cambridge, UK

Thesis Reader .....

Tommi S. Jaakkola

Associate Professor of Electrical Engineering and Computer Science

Massachusetts Institute of Technology

Thesis Reader .....

Zoubin Ghahramani

Reader in Machine Learning

Gatsby Computational Neuroscience Unit

University College London

Associate Research Professor of Computer Science

Carnegie Mellon University

Acknowledgments

I would like to express my gratitude to all those who helped me complete this thesis. First and foremost, I would like to offer special thanks to my advisor, Rosalind W. Picard, for not only giving me sage advice but also providing me with the freedom to explore a range of research projects. Her constant support and encouragement have pushed me to a place I could not have otherwise reached.

I am deeply indebted to Thomas P. Minka. I feel so lucky that I became his officemate at MIT a few years ago. Since then, his stimulating suggestions have been an invaluable resource for me. I have benefited from his keen insights, logical reasoning, and movie knowledge. I especially appreciate his delaying dinners because of our discussions after he moved to the "other" Cambridge in United Kingdom.

I also give warm thanks to my thesis committee members Tommi Jaakkola and Zoubin Ghahramani. I have enjoyed attending Tommi's machine learning classes and reading group. It was in his reading group that I first came across conditional random fields. This knowledge later turned out to be very useful for my internship at Microsoft Research. Tommi also served on my general examination committee with Josh Tenenbaum.

Zoubin generously hosted my visits to the Gatsby unit at University of College London and to Carnegie Mellon University. These trips allowed me to exchange ideas with researchers at different universities. In London, I finished my work with Zoubin on Bayesian feature selection and had interesting discussions with David MacKay.

I was fortunate to spend an autumn working with Martin Szummer and Christopher M. Bishop on ink recognition in Cambridge, UK. This project inspired me to start developing Bayesian conditional random fields. Besides Martin and Chris, I also want to thank Ralf Herbrich, Thore Graepel, Andrew Blake, and Michel Gangnet at Microsoft Research.

I give thanks to my fellow graduate students in Roz' group, especially Ashish Kapoor, Carson Reynolds, Karen Liu, Phil Davis, Win Burleson, and Raul Fernandez for the help they have given me. Moreover, I extend my thanks to the researchers and students at the Gatsby unit, especially Chu Wei, Jaz Kandola, Fernando Prez-Cruz, Ed Snelson, Iain Murray, and Katherine Heller.

Yufei Huang introduced me to adaptive signal detection and other interesting wireless communication problems; Peter Gorniak and Brian Whitman gave me the access to their

fast computer clusters; Fei Sha and Andrew McCallum offered me useful Java code and experimental datasets, respectively. Thanks to all for helping me on the thesis in one way or another.

I must thank all my Chinese friends with whom I have shared many good times and from whom I have gained strength during difficult times. There are so many of them I wish to thank but hopefully they know who they are. To name just a few: Wei Chai, Pengkai Pan, Yicong Li, Wei Luo, Junxu Zhang, and Ying He.

Finally, I would like to thank Jing Wu and my parents for their patient love and consistent support. I am forever grateful.

Contents

Chapter 1

Introduction

This thesis proposes efficient Bayesian inference and learning algorithms for graphical models. Inference problems are widely encountered in many important real-world applications, including human behavior recognition, audio and video processing, economics, and robotics. For instance, for wireless communications, given noisy received signals, how can a mobile device infer the original transmitted symbols from the noisy received signals and estimate the effect of the constantly changing environment? For these applications, a system of interest could be first modeled by a dynamic state-space model or a general graphical model. Then, given data and the constructed model, we can solve the problems encountered in these applications by estimating probability distributions of hidden variables of the model.

While for inference problems we know model parameters beforehand, for learning problems we estimate the parameters during training. A classical learning problem is feature selection, for which we try to classify the data in the presence of irrelevant features; one example is the classification of gene expression datasets into different categories. The challenge is that probably only a small subset of thousands of genes is relevant to the classification task. Another example is to classify relational data such as web pages, which are linked to each other. We can solve the above problems by first modeling the data using graphical models, learning the posterior distribution of model parameters, and then using the learned parameters for testing.

1.1 Bayesian inference and learning for graphical models

Compared with other inference and learning techniques, Bayesian approaches have a number of unique features. Bayesian approaches can seamlessly combine prior knowledge with data, provide error bars or confidence intervals for estimation and prediction, offer mechanisms for model and feature selection as well as hyperparameter tuning, and avoid overfitting by averaging prediction over the posterior distribution of model parameters.

One of the major barriers for Bayesian approaches to be used in practice is computational complexity. Monte Carlo methods have been widely used for Bayesian inference and learning (MacKay, 1998; Liu, 2001). Although flexible to use for almost any Bayesian model and achieving accurate results given a large amount of samples, Monte Carlo methods tend to be computationally expensive. The underlying reason for their computational inefficiency is that Monte Carlo methods are randomized algorithms, often wasting many samples in regions with low probabilities. Deterministic approximation methods have been developed to achieve much higher computational efficiency than Monte Carlo methods. Laplace's method approximates the posterior distribution by a Gaussian distribution based on the mode and the Hessian of the posterior (Bishop, 1995). Variational methods convert inference or learning into an optimization problem using lower- or upper-bounding techniques (Jordan et al., 1998; Beal, 2003). Expectation propagation (EP), proposed by Minka (2001), turns assumed density filtering into a smoothing method. On the problems investigated by Minka (2001), EP achieves more accurate estimation with comparable or less complexity than the other deterministic approximation techniques.

This thesis extends EP to broaden its applicability, enhance its efficiency, and improve its accuracy for graphical models. The goal of this thesis is to demonstrate that Bayesian inference and learning, based on the extensions of EP, can be both theoretically elegant and practically efficient. This thesis deals with four types of graphical models, as shown in Figure 1-1. In this figure, the shaded nodes represent observed variables and the unshaded represent latent variables. The two graphs at the top of the figures represent generative models, which model sampling processes of the observations; the two graphs at the bottom represent conditional models, where we try to infer the posterior distribution of model parameters $\mathbf{w}$ conditional on the observations.

Specifically, the left upper graph is a Bayesian network, a directed and generative graph-

ical model; the right upper graph is a Markov random field, an undirected and generative graph model; the left bottom graph is a traditional classification model, where we assume data labels are independent of each other given the inputs and model parameters; the right bottom graph is a conditional random field, where we capture the relation between data labels using undirected edges in the graph.

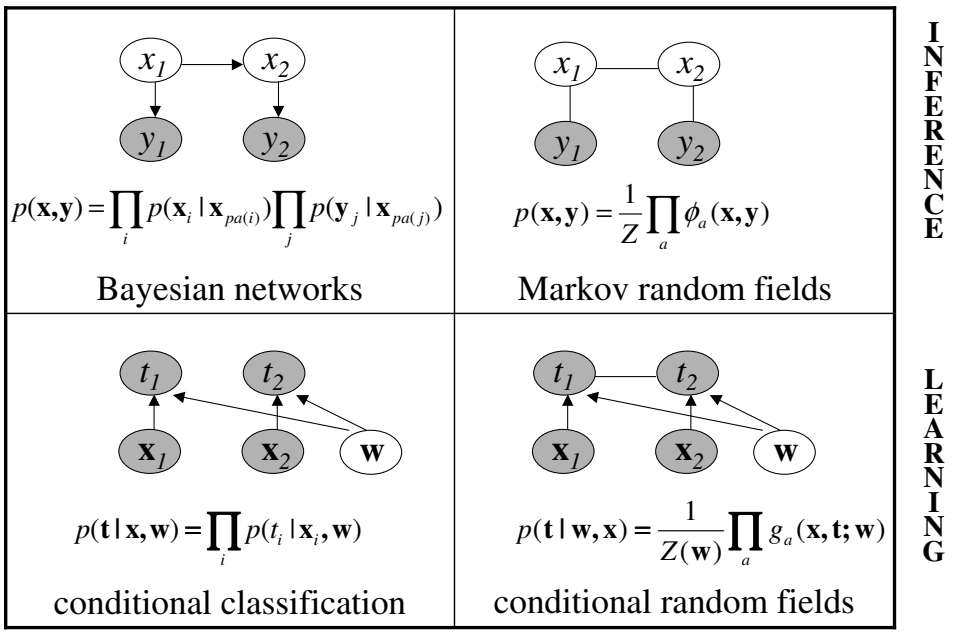


Figure 1-1: Different types of graphical models. The shaded nodes represent observed variables.

1.2 Extensions to expectation propagation (EP)

Corresponding to the four types of graphical models, the thesis proposes two inference algorithms for the generative models at the top and two learning algorithms for the conditional models at the bottom. Specifically, the thesis extends EP for the following tasks:

Task: We often encounter online estimation for dynamic systems with nonlinear and non-Gaussian likelihoods. One example is wireless signal detection, which we will consider in this thesis. As a batch-smoothing method, EP is computationally expensive for online estimation.

Extension: The thesis proposes window-based EP smoothing such that iterative EP refinement can be used for online applications, which are modeled by dynamic systems

with nonlinear and non-Gaussian likelihood. Window-based EP smoothing can be viewed as a trade-off between batch EP smoothing and assumed density filtering. By controlling the window length for window-based EP smoothing, we can easily govern computational complexity.

Task: Random fields can be used for many applications, such as computer vision and sensor networks. For random fields with cycles, it is expensive to compute exact marginal distributions. How to efficiently approximate marginal distributions has become an important research topic. One popular approach is belief propagation (BP) (Pearl, 1988; K. P. Murphy, 1999). From the EP perspective, BP uses factorized approximation structures. To extend BP, we can use tree-structured EP approximation to capture long range correlations between variables. However, we need a clever data structure to make the algorithm efficient. Otherwise, a straightforward implementation of globally structured EP approximation will lead to global message propagation at each EP update, which is expensive both in computational time and memory size.

Extension: This thesis develops a new method that combines tree-structured EP approximation with junction tree representation for inference on loopy graphical models. With the combination, a local consistent junction tree is computed during updates, and a global consistent junction tree is achieved only at the end of updates. Local propagation enhances algorithmic efficiency both in time and memory.

Task: EP has been previously used for classification with Bayes point machine (BPM) models (Minka, 2001). However, in many real-world classification problems the input contains a large number of potentially irrelevant features, which can degenerate the prediction performance of BPM classification.

Extension: The thesis proposes *predictive* automatic relevance determination (ARD), which combines EP with ARD, for determining the relevance of input features, in order to improve EP classification accuracy. ARD is one of the most successful Bayesian methods for feature selection and sparse learning. ARD finds the relevance of features by optimizing the model marginal likelihood, also known as the evidence. The thesis shows that this can lead to overfitting. To address this problem, predictive-ARD estimates the predictive performance of the classifier. While the actual leave-one-out

predictive performance is generally very costly to compute, EP provides an estimate of this predictive performance as a side-effect of its iterations. The thesis exploits this property for feature selection as well as for data point selection in a sparse Bayesian kernel classifier. Moreover, this thesis uses sequential optimization to avoid the computation involving the full covariance matrix of parameters, increasing algorithmic efficiency.

Task: Traditional classification methods often assume data are independent of each other, while a lot of real-world data, such as sequences, images, or webs, have interdependent relations and complex structures. The thesis generalizes traditional Bayesian classification and proposes Bayesian conditional random fields (BCRFs) for classifying interdependent and structured data. BCRFs are a Bayesian approach to training and inference with conditional random fields, which were previously trained by maximum likelihood (ML) and maximum a posteriori (MAP) approaches (Lafferty et al., 2001). Unlike ML and MAP approaches, BCRFs estimate the posterior distribution of the model parameters during training, and average the prediction over this posterior during inference. This new approach avoids the problem of overfitting and offers the full advantages of a Bayesian treatment. However, BCRF training is not an easy task. The main difficulty of BCRF training comes from the partition function: it is a complicated function and appears in the denominator of the likelihood. While traditional EP and variational methods can not be directly applied, because of the presence of partition functions, the power EP (PEP) method (Minka, 2004) can be used for BCRF training. However, PEP can lead to convergence problems.

Extension: To solve the convergence problems, this thesis proposes transformed PEP to incorporate partition functions in BCRF training. Moreover, for algorithmic stability and accuracy, BCRFs flatten approximation structures to avoid two-level approximations. Low-rank matrix operations are also explored to reduce computational complexity.

1.3 Thesis overview

The succeeding chapters of this thesis are organized as follows. Chapter 2 presents the new window-based EP smoothing algorithm, and compares it with sequential Monte Carlo

methods for wireless signal detection in flat-fading channels. Chapter 3 presents the new inference method on loopy graphs that combines tree-structured EP approximations with the junction tree representation. In Chapter 4, we propose predictive-ARD for feature selection and sparse kernel learning with applications for gene expression classification. For classifying relational data, Chapter 5 describes Bayesian conditional random fields (BCRFs), a novel Bayesian approach for classifying interdependent and structured data, and applies BCRFs to natural language processing and hand-drawn diagram recognition. Finally, Chapter 6 concludes the thesis and presents future work.

Chapter 2

Extending EP for dynamic models with nonlinear/non-Gaussian likelihoods

2.1 Introduction

This chapter[1] first presents a batch EP smoothing algorithm on dynamic models with nonlinear/non-Gaussian likelihoods. Then this chapter presents a window-based EP smoothing algorithm, which extends batch EP smoothing for online applications. Finally, it compares window-based EP smoothing with sequential Monte Carlo methods for wireless signal detection in flat-fading channels.

2.2 Approximate inference on dynamic models with nonlinear/non-Gaussian likelihoods

For discrete or linear Gaussian dynamic models, we have efficient and elegant inference algorithms such as the forward-backward algorithm as well as Kalman filtering and smoothing (Minka, 1998). However, we encounter in practice many dynamic models with continuous state variables and nonlinear/non-Gaussian likelihoods. For these kinds of dynamic models, more complicated approaches are needed to do the inference. Most of these ap-

[1]This chapter includes joint work with T.P. Minka (Qi & Minka, 2003).

proaches can be categorized into two classes: deterministic approximation methods and Monte Carlo methods.

As a popular deterministic approximation method, extended Kalman filtering and smoothing (Maybeck, 1979; Welch & Bishop, 2002) linearizes the process and measurement equations by a Taylor expansion about the current state estimate. The noise variance in the equations is not changed, i.e. the additional error due to linearization is not modeled. After linearization, the classical Kalman filter and smoother are applied. Moreover, variational methods have been applied to dynamic models, where the "exclusive" KL divergence $KL(q||p)$ between the approximate posterior distribution q and the true posterior p is minimized by a lower-bound maximization (Ghahramani & Beal, 2000). Assumed-density filtering sequentially minimizes the "inclusive" KL divergence $KL(p||q)$ between the true posterior p and the approximate posterior distribution q, and has achieved superior results over classical nonlinear filtering methods, e.g., extended Kalman filtering (Maybeck, 1979). For the difference between "inclusive" and "exclusive" KL minimization, we refer the readers to Frey et al. (2000).

Monte Carlo methods can generally achieve more accurate inference results than deterministic approximation methods, once having drawn a sufficiently large amount of samples. Markov Chain Monte Carlo methods, including Gibbs sampling and Metropolis-Hastings, have been applied to dynamic models to achieve accurate results (Cargnoni et al., 1997; Tanizaki, 2000). Also, resampled sequential Monte Carlo, i.e., particle filtering and smoothing has been used to explore the Markovian property of dynamic models for efficient inference (Doucet et al., 2001; Fong et al., 2001). Since the inference accuracy heavily depends on the number of samples, Monte Carlo methods are generally much slower than deterministic approximation methods.

As shown by Minka (2001) on a variety of problems, EP achieves more accurate inference than variational methods and obtains more efficient inference than Monte Carlo methods. These results motivated us to develop EP batch smoothing algorithm for dynamic systems for a good trade-off between accuracy and efficiency.

2.3 Signal detection problem

In this section, we consider a specific real-world application of dynamic systems, wireless signal detection in flat-fading channels. A wireless communication system with a fading channel can be modeled as (Chen et al., 2000)

$$y_t = s_t \alpha_t + w_t, \qquad t = 0, 1, \ldots \tag{2.1}$$

where y_t, s_t, α_t and w_t are the received signal, the transmitted symbol, the fading channel coefficient, and the complex Gaussian noise $\mathcal{N}_c(0, \sigma^2)$, respectively. The symbols $\{s_t\}$ take values from a finite alphabet $\mathcal{A} = \{a_i\}_{i=1}^{M}$. The fading coefficients $\{\alpha_t\}$ can be modeled by a complex autoregressive moving-average (ARMA) process as follows:

$$\alpha_t = \sum_{i=0}^{\rho} \theta_i v_{t-i} - \sum_{i=1}^{\rho} \phi_i \alpha_{t-i}$$

where $\Theta = \{\theta_t\}$ and $\Phi = \{\phi_t\}$ are the ARMA coefficients, and v_t is the white complex Gaussian noise with unit variance.

For simplicity, we consider only the uncoded case, where each α_t in $\mathcal{A}$ has an equal prior probability. Define $\mathbf{x}_t$ in terms of α_t, we can rewrite the wireless communication system as a state-space model:

$$\mathbf{x}_t = \mathbf{F}\mathbf{x}_{t-1} + \mathbf{g}_t v_t \tag{2.2}$$

$$y_t = s_t h^H \mathbf{x}_t + w_t \tag{2.3}$$

where

$$\mathbf{F} = \begin{pmatrix} -\phi_1 & -\phi_2 & \ldots & -\phi_\rho & 0 \\ 1 & 0 & \ldots & 0 & 0 \\ 0 & 1 & \ldots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \ldots & 1 & 0 \end{pmatrix}, \qquad \mathbf{g} = \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix},$$

$$h = [\theta_0, \theta_1, \ldots, \theta_\rho]^H,$$

and the dimension of $\mathbf{x}$ is $d = \rho + 1$. Note that H means hermitian transpose. We can represent (2.2) and (2.3) by the following graphical model:

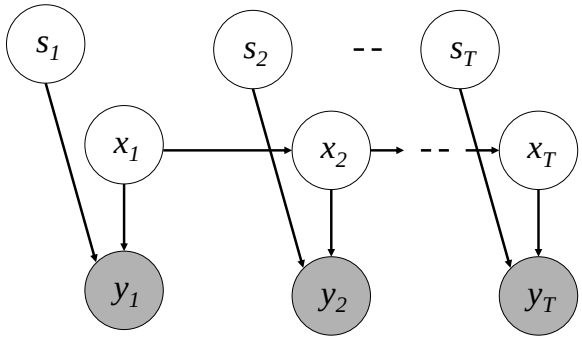


Figure 2-1: Graphical model for adaptive signal detection. The shaded nodes in the graph are observed.

The signal detection problem can then be formulated as an inference problem in the dynamic system defined by (2.2) and (2.3). We address this problem using a Bayesian approach. Specifically, for filtering, we update the posterior distribution $p(s_t, \mathbf{x}_t|y_{1:t})$ based on the observations from the beginning to the current time t, i.e., $y_{1:t} = [y_1, \ldots, y_t]$. For smoothing, we compute the posterior $p(s_t, \mathbf{x}_t|y_{1:T})$ based on the whole observation sequence, i.e., $y_{1:T} = [y_1, \ldots, y_T]$, where T is the length of the observation sequence. Since smoothing uses more information from the observations than filtering, smoothing generally achieves more accurate estimation than filtering.

If a dynamic system is linear and has Gaussian noises, then we can use Kalman filtering and smoothing to efficiently compute the posterior distribution $p(s_t, \mathbf{x}_t|y_{1:T})$. Otherwise, we need to resort to other advanced techniques to approximate the posterior. Instead of using the sequential Monte Carlo method (Chen et al., 2000; Wang et al., 2002), we utilize expectation propagation to efficiently approximate the posterior $p(s_t, \mathbf{x}_t|y_{1:t+L-1})$ for smoothing.

2.4 Smoothing for hybrid dynamic models

In this section, we develop the expectation propagation algorithm for the hybrid dynamic system defined by (2.2) and (2.3). Expectation propagation exploits the fact that the likelihood is a product of simple terms. If we approximate each of these terms well, we can get a good approximation to the posterior. Expectation propagation chooses each

approximation such that the posterior using the term exactly and the posterior using the term approximately are close in KL-divergence. This gives a system of coupled equations for the approximations which are iterated to reach a fixed-point.

Specifically, the exact posterior distribution is proportional to a product of observation densities and transition densities as follows:

$$o(s_t, \mathbf{x}_t) = p(y_t|s_t, \mathbf{x}_t) \tag{2.4}$$

$$g_t(s_t, \mathbf{x}_t, s_{t+1}, \mathbf{x}_{t+1}) = p(s_{t+1}, \mathbf{x}_{t+1}|s_t, \mathbf{x}_t) \tag{2.5}$$

$$= p(\mathbf{x}_{t+1}|\mathbf{x}_t)p(s_{t+1}) \tag{2.6}$$

$$p(s_{1:T}, \mathbf{x}_{1:T}|y_{1:T}) \propto p(s_1, \mathbf{x}_1)o(s_1, \mathbf{x}_1)\cdot \cdot \prod_{i=1}^{T} g_{t-1}(s_{t-1}, \mathbf{x}_{t-1}, s_t, \mathbf{x}_t)o(s_t, \mathbf{x}_t) \tag{2.7}$$

Equation (2.6) holds because each s_t is independent of the others at different times in the dynamic model.

Then we approximate the posterior by the product of the independent terms:

$$p(s_{1:T}, \mathbf{x}_{1:T}|y_{1:y_T}) \approx \prod_{i=1}^{T} q(s_i, \mathbf{x}_i) \tag{2.8}$$

where $q(s_i, \mathbf{x}_i)$ can be interpreted as the approximation of the state posterior, i.e., the state belief. We give more details on how to approximate state belief later.

Correspondingly, the terms $o(s_t, \mathbf{x}_t)$ and $g_t(s_t, \mathbf{x}_t, s_{t+1}, \mathbf{x}_{t+1})$ in (2.7) are approximated by $\tilde{o}(s_t, \mathbf{x}_t)$ and $\tilde{g}_t(s_t, \mathbf{x}_t, s_{t+1}, \mathbf{x}_{t+1})$. To decouple the states in (2.7), we set

$$\tilde{g}_t(s_t, \mathbf{x}_t, s_{t+1}, \mathbf{x}_{t+1}) = \tilde{g}_t(s_t, \mathbf{x}_t)\tilde{g}_t(s_{t+1}, \mathbf{x}_{t+1}).$$

We can interpret these approximation terms as messages that propagate in the dynamic system: $\tilde{o}(s_t, \mathbf{x}_t)$ is an *observation message* from y_t to $(s_t, \mathbf{x}_t)$, $\tilde{g}_t(s_{t+1}, \mathbf{x}_{t+1})$ a *forward message* from $(s_t, \mathbf{x}_t)$ to $(s_{t+1}, \mathbf{x}_{t+1})$, and $\tilde{g}_t(s_t, \mathbf{x}_t)$ a *backward message* from $(s_{t+1}, \mathbf{x}_{t+1})$ to $(s_t, \mathbf{x}_t)$. These messages are illustrated in the following:

After all of these approximations are made, each approximate state posterior $q(s_t, \mathbf{x}_t)$

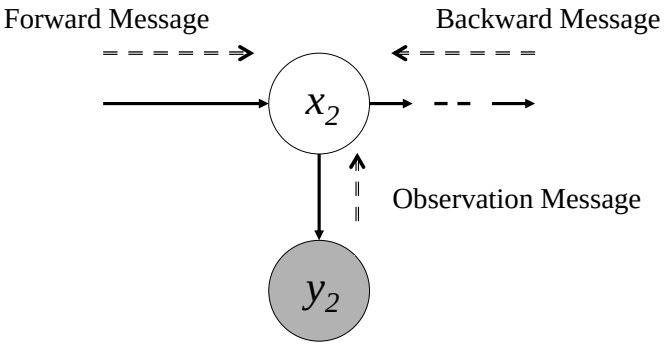


Figure 2-2: Messages obtained from EP approximations in a dynamic model

is a product of three messages:

$$\begin{aligned} q(s_t, \mathbf{x}_t) &\propto \tilde{g}_{t-1}(s_t, \mathbf{x}_t)\tilde{o}(s_t, \mathbf{x}_t)\tilde{g}_t(s_t, \mathbf{x}_t) \\ &= (\text{forward})(\text{observation})(\text{backward}) \end{aligned}$$

In the following sections, we describe how to compute and incorporate these messages.

2.4.1 Moment matching and observation message update

First, consider how to update the state belief $q(s_t, \mathbf{x}_t)$ using the observation data and, correspondingly, how to generate the observation message .

Denote by $q^{\backslash o}(s_t, \mathbf{x}_t)$ the belief of the state $(s_t, \mathbf{x}_t)$ before incorporating the observation message. For simplicity, we assume s_t and $\mathbf{x}_t$ are both statistically independent and in the exponential family, so that

$$\begin{aligned} q^{\backslash o}(s_t, \mathbf{x}_t) &= q^{\backslash o}(s_t)q^{\backslash o}(\mathbf{x}_t) \\ q^{\backslash o}(s_t) &\sim \text{Discrete}(p_{t,1}^{\backslash o}, p_{t,2}^{\backslash o} \ldots, p_{t,M}^{\backslash o}) \\ q^{\backslash o}(\mathbf{x}_t) &\sim \mathcal{N}_c(\mathbf{m}_t^{\backslash o}, V_t^{\backslash o}) \end{aligned}$$

where $p_{t,i}^{\backslash o}$ is shorthand of $q^{\backslash o}(s_t = a_i)$.

Given $q^{\backslash o}(s_t)q^{\backslash o}(\mathbf{x}_t)$ and $o(s_t, \mathbf{x}_t)$, we can obtain the posterior $\hat{q}(s_t, \mathbf{x}_t)$:

$$\hat{q}(s_t, \mathbf{x}_t) = \frac{o(s_t, \mathbf{x}_t)q^{\backslash o}(s_t, \mathbf{x}_t)}{\int_{s_t, \mathbf{x}_t} o(s_t, \mathbf{x}_t)q^{\backslash o}(s_t, \mathbf{x}_t)} \tag{2.9}$$

Define $Z = \int_{s_t,\mathbf{x}_t} o(s_t, \mathbf{x}_t) q^{\backslash \mathbf{o}}(s_t, \mathbf{x}_t)$. Then it follows that

$$\begin{aligned} Z &= \sum_{s_t \in \mathcal{A}} q^{\backslash o}(s_t) \int \mathcal{N}_c(y_t | s_t h^H \mathbf{x}_t, \sigma^2) \mathcal{N}_c(\mathbf{x}_t | \mathbf{m}_t^{\backslash o}, V_t^{\backslash o}) \mathrm{d}\mathbf{x}_t \\ &= \sum_{s_t \in \mathcal{A}} q^{\backslash o}(s_t) \mathcal{N}(y_t | \mathbf{m}_{y_t}, V_{y_t}) \end{aligned} \tag{2.10}$$

where

$$\mathbf{m}_{y_t} = s_t h^H \mathbf{m}_t^{\backslash o}, \tag{2.11}$$

$$V_{y_t} = s_t h^H V_t^{\backslash o} h s_t^H + \sigma^2, \tag{2.12}$$

and $\mathcal{N}_c(\cdot | \mathbf{m}_t^{\backslash o}, V_t^{\backslash o})$ is the probability density function of a complex Gaussian with mean of $\mathbf{m}_t^{\backslash o}$ and variance of $V_t^{\backslash o}$.

However, we cannot keep $\hat{q}(s_t, \mathbf{x}_t)$ as the new belief of $(s_t, \mathbf{x}_t)$ for message propagation. The reason is that $\hat{q}(s_t, \mathbf{x}_t)$ is not in the exponential family and, therefore, we cannot keep updating the state belief in the dynamic model analytically and efficiently. To solve this problem, we project $\hat{q}(s_t, \mathbf{x}_t)$ into an approximate distribution $q(s_t, \mathbf{x}_t)$ in the exponential family:

$$q(s_t, \mathbf{x}_t) = q(\mathbf{x}_t) q(s_t) \tag{2.13}$$

$$q(\mathbf{x}_t) \sim \mathcal{N}_c(\mathbf{m}_t, V_t) \tag{2.14}$$

$$q(s_t) \sim \text{Discrete}(p_{t,1}, p_{t,2}, \ldots, p_{t,M}) \tag{2.15}$$

The projection criterion is to minimize the KL divergence between $\hat{q}$ and q, $\text{KL}(\hat{q}||q)$. This step is the same as assumed-density filtering (Kushner & Budhiraja, 2000; Boyen & Koller, 1998).

For this minimization, we match the moments between $\hat{q}$ and q:

$$p_{t,i} = \frac{p_{t,i}^{\backslash o} \mathcal{N}(y_t | a_i h^H \mathbf{m}_t^{\backslash o}, V_{y_t})}{Z} \tag{2.16}$$

$$\mathbf{m}_t = \frac{\sum_{s_t \in \mathcal{A}} q^{\backslash o}(s_t) \mathcal{N}(y_t | \mathbf{m}_{y_t}, V_{y_t}) \mathbf{m}_{x_t | y_t}}{Z} \tag{2.17}$$

$$V_t = V_{x_t | y_t} - \mathbf{m}_t \mathbf{m}_t^H +$$

$$+ \sum_{s_t \in \mathcal{A}} q^{\setminus o}(s_t) \mathcal{N}(y_t | \mathbf{m}_{y_t}, V_{y_t}) \mathbf{m}_{x_t|y_t} \mathbf{m}_{x_t|y_t}^H / Z \tag{2.18}$$

where

$$\mathbf{m}_{x_t|y_t} = \mathbf{m}_t^{\setminus o} + K_{s_t}(y_t - s_t h^H \mathbf{m}_t^{\setminus o}) \tag{2.19}$$

$$V_{x_t|y_t} = V_t^{\setminus o} - K_{s_t} s_t h^H V_t^{\setminus o} \tag{2.20}$$

$$K_{s_t} = V_t^{\setminus o} h s_t^H V_{y_t}^{-1} \tag{2.21}$$

Then we compute the observation message as follows:

$$\tilde{o}(s_t, \mathbf{x}_t) = Z \frac{q(s_t, \mathbf{x}_t)}{q^{\setminus o}(s_t, \mathbf{x}_t)}. \tag{2.22}$$

It follows that

$$\tilde{o}(s_t, \mathbf{x}_t) = \tilde{o}(\mathbf{x}_t)\tilde{o}(s_t) \tag{2.23}$$

$$\tilde{o}(\mathbf{x}_t) \sim \mathcal{N}(\tilde{\boldsymbol{\mu}}_t, \tilde{\Lambda}_t) \tag{2.24}$$

$$\tilde{o}(s_t) \sim \text{Discrete}(r_{t,1}, r_{t,2}, \ldots, r_{t,M}) \tag{2.25}$$

where

$$\tilde{\boldsymbol{\mu}}_t = \tilde{\Lambda}_t V_t^{-1} \mathbf{m}_t - \tilde{\Lambda}_t (V_t^{\setminus o})^{-1} \mathbf{m}_t^{\setminus o} \tag{2.26}$$

$$\tilde{\Lambda}_t = (V_t^{-1} - (V_t^{\setminus o})^{-1})^{-1} \tag{2.27}$$

Note that the observation message is not necessarily a valid probability distribution. For example, $\tilde{\Lambda}_t$ may not be positive definite, or some of its elements may even go to infinity. To avoid numerical problems, we transform $\tilde{\boldsymbol{\mu}}_t$ and $\tilde{\Lambda}_t$ to the natural parameterization of the exponential family:

$$\boldsymbol{\mu}_t = \tilde{\Lambda}_t^{-1} \tilde{\boldsymbol{\mu}}_t = V_t^{-1} \mathbf{m}_t - (V_t^{\setminus o})^{-1} \mathbf{m}_t^{\setminus o} \tag{2.28}$$

$$\Lambda_t = \tilde{\Lambda}_t^{-1} = V_t^{-1} - (V_t^{\setminus o})^{-1} \tag{2.29}$$

$$r_{t,j} = \frac{p_{t,j}}{p_{t,j}^{\setminus o}} \qquad \text{for } j = 1, \ldots, M \tag{2.30}$$

Since Λ_t is often near singular, inverting Λ_t loses numerical accuracy. Using the natural parameters, we keep accuracy by avoiding to invert Λ_t.

2.4.2 Incorporating forward, observation, and backward messages

In this section, we describe how to incorporate the messages to update $q(s_t, \mathbf{x}_t)$. Because each s_t is independent of the others at different times, we consider only the belief update for $\mathbf{x}_t$ when incorporating forward and backward messages. Since all the marginal messages and belief which are related to $\mathbf{x}_t$ are Gaussians, we can efficiently update them.

1. Compute and incorporate the forward message $\tilde{g}_{t-1}(\mathbf{x}_t)$. Set $q^{\backslash o}(\mathbf{x}_t) = \tilde{g}_{t-1}(\mathbf{x}_t)$ such that

$$\mathbf{m}_t^{\backslash o} = \mathbf{F}\mathbf{m}_{t-1} \tag{2.31}$$

$$V_t^{\backslash o} = \mathbf{F}V_{t-1}\mathbf{F}^H + \mathbf{g}\mathbf{g}^H. \tag{2.32}$$

   $\mathbf{m}_0$ and V_0 are chosen as the prior. Also, set $P_t = V_t^{\backslash o}$, which will be used later when incorporating the backward message.

2. Incorporate the observation message $\tilde{o}(s_t, \mathbf{x}_t)$. In the previous section, we compute $\tilde{o}(s_t, \mathbf{x}_t)$ based on the update of $q(s_t, \mathbf{x}_t)$. On the other hand, given $\tilde{o}(s_t, \mathbf{x}_t)$ and $q_t^{\backslash o}(\mathbf{x}_t)$, we can update $q(s_t, \mathbf{x}_t)$ by rewriting (2.28) to (2.30) as follows:

$$\mathbf{m}_t = V_t(\boldsymbol{\mu}_t + (V_t^{\backslash o})^{-1}\mathbf{m}_t^{\backslash o}) \tag{2.33}$$

$$V_t = (P_t^{-1} + \Lambda_t)^{-1} \tag{2.34}$$

$$p_{t,j} = r_{t,j}p_{t,j}^{\backslash o} \tag{2.35}$$

3. Incorporate the backward message $\tilde{g}_t(\mathbf{x}_t)$. Without explicitly computing the backward message $\tilde{g}_t(\mathbf{x}_t)$, we can directly incorporate $\tilde{g}_t(\mathbf{x}_t)$ into $q(\mathbf{x}_t)$ as Kalman smoothing:

$$\mathbf{J}_t = V_t^{\backslash b}F^H P_t^{-1} \tag{2.36}$$

$$\mathbf{m}_t = \mathbf{m}_t^{\backslash b} + \mathbf{J}_t(\mathbf{m}_{t+1} - \mathbf{F}\mathbf{m}_t^{\backslash b}) \tag{2.37}$$

$$V_t = V_t^{\backslash b} + \mathbf{J}_t(V_{t+1}\mathbf{J}_t^H - \mathbf{F}(V_t^{\backslash b})^H) \tag{2.38}$$

where ($\mathbf{m}_t^{\backslash b}$, $V_t^{\backslash b}$) and ($\mathbf{m}_t$, V_t) are the means and variances of the state belief before and after incorporating the backward message, respectively.

2.4.3 Algorithm summary

Given the knowledge of how to incorporate different messages, we are ready to construct the whole expectation propagation algorithm by establishing the iteration mechanism.

1. Initialize the parameters $p_{t,j}, \mathbf{m}_0, V_0, \boldsymbol{\mu}_t, \Lambda_t$.
2. Then loop $t = 1 : T$:
   (a) Set $q^{\backslash o}(\mathbf{x}_t)$ to the forward message from $\mathbf{x}_{t-1}$ via (2.31) and (2.32).
   (b) Update $q(s_t, \mathbf{x}_t)$ to match the moments of $o(s_t, \mathbf{x}_t)q^{\backslash o}(s_t, \mathbf{x}_t)$ via (2.16) to (2.18).
   (c) Compute $\tilde{o}(s_t, \mathbf{x}_t) \propto q(s_t, \mathbf{x}_t)/q^{\backslash o}(s_t, \mathbf{x}_t)$ via (2.28) to (2.30).
3. Loop by increasing i until i equals n, or the convergence has been achieved:
   (a) Loop $t = 1, \ldots, T$ (Skip on the first iteration)
      i. Set $q^{\backslash o}(\mathbf{x}_t)$ to the forward message from $\mathbf{x}_{t-1}$ via (2.31) and (2.32).
      ii. Set $q(s_t, \mathbf{x}_t)$ to the product $\tilde{o}(s_t, \mathbf{x}_t)q^{\backslash o}(s_t, \mathbf{x}_t)$ via (2.33) to (2.35).
   (b) Loop $t = T, \ldots, 1$
      i. Set $\mathbf{m}_t^{\backslash b} = \mathbf{m}_t$ and $V_t^{\backslash b} = V_t$
      ii. Update $q(s_t, \mathbf{x}_t)$ by incorporating the backward message via (2.36) to (2.38) when $t < T$.
      iii. Update $q(s_t, \mathbf{x}_t)$ and $\tilde{o}(s_t, \mathbf{x}_t)$ as follows:
         A. Delete the current observation message from $q(s_t, \mathbf{x}_t)$. This is an important step, in order to avoid double-counting the observation message $\tilde{o}(s_t, \mathbf{x}_t)$:

$$q^{\backslash o}(s_t, \mathbf{x}_t) \propto q(s_t, \mathbf{x}_t)/\tilde{o}(s_t, \mathbf{x}_t).$$

Then it follows that

$$\mathbf{m}_t^{\backslash o} = V_t^{\backslash o}(V_t^{-1}\mathbf{m}_t - \boldsymbol{\mu}_t), \tag{2.39}$$

$$V_t^{\backslash o} = (V_t^{-1} - \Lambda_t)^{-1}, \tag{2.40}$$

$$p_{t,j}^{\setminus o} = p_{t,j}/r_{t,j} \tag{2.41}$$

B. Update $q(s_t, \mathbf{x}_t)$ to match the moments of $o(s_t, \mathbf{x}_t)q^{\setminus o}(s_t, \mathbf{x}_t)$ via (2.16) to (2.18).

C. Compute via (2.28) to (2.30) $\tilde{o}(s_t, \mathbf{x}_t) \propto q(s_t, \mathbf{x}_t)/q^{\setminus o}(s_t, \mathbf{x}_t)$.

Note that for dynamic systems with nonlinear and non-Gaussian likelihoods different from the signal detection problem, the only thing we need to modify in the above description is the moment matching step, defined by (2.16) to (2.18). Based on a different likelihood, we will have a different moment matching step. When exact moments are difficult to compute, we can use Taylor expansions, unscented Kalman filtering (Wan et al., 2000), Monte Carlo, or quadrature methods to approximate the moment matching step.

Moreover, we want to highlight the difference between EP smoothing and traditional smoothing methods. For a linear Gaussian dynamic model or a discrete dynamic model, we will obtain the exact posterior of any state given the whole observation sequence after propagating all the forward, observation, and backward messages once. In other words, we need only one forward pass and one backward pass of the observation sequence. For a dynamic system with nonlinear and non-Gaussian likelihoods, extended Kalman smoothing, a traditional technique, linearizes the observation likelihood and then applies classical Kalman filtering and smoothing with only one forward pass and one backward pass. As described above, unlike the traditional techniques, EP smoothing will iterate the forward and backward passes until convergence. This iteration enables the refinement of observation approximations, which in turn leads to better approximation of the posterior distributions.

2.5 Window-based EP smoothing

For many real-world applications, including the signal detection problem, we need online processing of the data. This need cannot currently be satisfied by batch EP smoothing, which uses the entire observation sequence and therefore is computationally expensive.

To address this problem, we propose window-based EP smoothing as an alternative to batch EP smoothing. Window-based EP smoothing finds a trade-off between assumed-density filtering and batch EP smoothing. Instead of smoothing over the entire observation sequence, we use a sliding window with length L to approximate the posterior $p(s_t, \mathbf{x}_t|y_{1:t+\delta})$

based on the observations $y_{1:t+\delta} = [y_1, \ldots, y_{t+\delta}]$, where δ controls the delay for online estimation. Specifically, we first run ADF filtering from time t to $t+\delta$, perform EP smoothing from $t+\delta$ to $t+\delta-L$, and then run ADF filtering from $t+\delta-L$ to $t+\delta$. We iterate the filtering and smoothing in the sliding window until a fixed number of iterations or the convergence has achieved. Essentially, window-based EP smoothing is a rescheduling of EP update orders for the online-estimation purpose. The difference between ADF, batch EP, and window-based EP is illustrated in Figure 2-3.

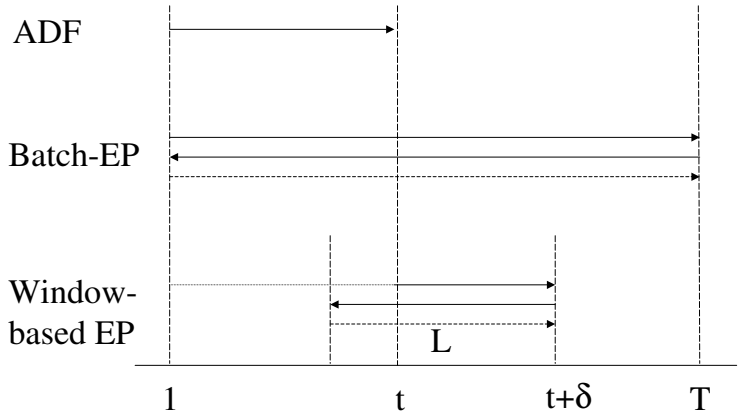


Figure 2-3: Illustration of ADF, batch EP, and window-based EP. ADF sequentially processes the observation sequence to the current time t; batch EP smoothing uses the entire observation sequence from the beginning at time 1 to the end at time T; window-based EP uses the previous approximate posterior at time t as a prior distribution for filtering and performs smoothing in a sliding window with length L.

2.6 Computational complexity

In this section, we compare the efficiency of window-based EP smoothing with sequential Monte Carlo methods. For window-based EP smoothing, the total computation time of incorporating the forward and observation messages via (2.31) to (2.35) is $O(d^2)$ (d is the dimension of $\mathbf{x}_t$), same as the cost of one-step Kalman filtering; incorporating the backward message via (2.36) to (2.38) takes $O(d^2)$ as Kalman smoothing; and finally updating $q(s_t, \mathbf{x}_t)$ and $\tilde{o}(s_t, \mathbf{x}_t)$ in step 3(b)iii costs $O(Md^2)$, M times as the cost of Kalman filtering, where M is the size of alphabet for symbols. Furthermore, since in general the estimation accuracy is not increasing after a few propagation iterations, the needed number of EP iterations n is small. In our experiments, we set $n = 5$. In sum, given the length of the smoothing window L, the size of alphabet for symbols M, and the number of EP iterations n, the computation

takes $O(nMLd^2)$, same as nML times the cost of one-step Kalman filtering and smoothing.

In contrast, if we use m samples in a stochastic mixture of Kalman filters, also known as Rao-blackwellised particle smoothers, it takes $O(mMd^2)$ for a one-step update, and $O(mM^Ld^2)$ for L step smoothing, which is mM^L times the cost of one-step Kalman filtering and smoothing (Chen et al., 2000). Another version of efficient particle smoothers (Fong et al., 2001), which is applied to audio processing, takes $O(mkMLd^2)$ where m and k are the numbers of forward and backward samples. To achieve accurate estimation, sequential Monte Carlo methods generally need a large number of samples, such that both m and k take large values.

Clearly, the computational cost of sequential Monte Carlo methods either explodes exponentially with the length of the sliding window, or increases at a rate proportional to the product of the number of forward and backward samples, while the cost of window-based EP smoothing increases only linearly with the window length.

2.7 Experimental results on wireless signal detection

First, we apply the proposed window-based EP smoothing algorithm to the signal detection problem. The flat-fading channels are defined in (2.2) and(2.3), and the proposed algorithm decodes the symbols s_t as

$$\hat{s}_t = \left\{ a_i | \arg\max_i \{\hat{p}_{t,i}\} \right\}. \tag{2.42}$$

where $\hat{p}_{t,i}$ is obtained after the convergence of the expectation propagation algorithm.

We demonstrate the high performance of the window-based EP receiver in a flat-fading channel with different signal noise ratios. We model the fading coefficients $\{\alpha_t\}$ by the following ARMA(3,3) model, as in (Chen et al., 2000): $\Phi = [-2.37409 \quad 1.92936 \quad -0.53208]$, $\Theta = 0.01 \times [0.89409 \quad 2.68227 \quad 2.68227 \quad 0.89409]$, $v_t \sim \mathcal{N}_c(0, 1)$. With these parameters, we have $\mathrm{Var}\{\alpha_t\} = 1$. BPSK modulation is employed; that is, $s_t \in \{1, -1\}$. In addition, differential encoding and decoding are employed to resolve the phase ambiguity.

We test the window-based EP receiver with different window lengths $L = 1, 2, 4$, with $0, 1, 3$ overlap points, respectively. In other words, the estimation time delay δ equals 0,1,and 3, respectively. Moreover, we run the batch EP receiver with smoothing over the entire data sequence.

For comparison, we test a genie-aided lower bound and a differential detector. For the genie-aided detection, an additional observation is provided, which is another transmitted signal where the symbol is always 1, i.e., $\tilde{y}_t = \alpha_t + w_t$. The receiver employs a Kalman filter to estimate the posterior mean $\hat{\alpha}_t$ of the fading process, based on the new observation sequence $\{\tilde{y}_t\}$. The symbols are then demodulated according to $\hat{s}_t = \text{sign}(\mathcal{R}\{\hat{\alpha}_t^\star y_t\})$ where $\star$ means conjugate. By obtaining the extra information from the genie, this detector is expected to achieve accurate detection results. For the differential detection, no attempt is made for channel estimation. It simply detects the phase difference between two consecutive observations y_{t-1} and y_t: $\hat{s}_t = \text{sign}(\mathcal{R}\{\hat{y}_t{}^\star y_{t-1}\})$.

We run these detectors on 50,000 received signals multiple times. Each time, we randomly synthesize a new symbol sequence and a new observation sequence according to (2.2) and (2.3). The signal-noise ratio (SNR), defined as $10\log_{10}(\text{Var}\{\alpha_t\}/\text{Var}\{w_t\})$, increases each time. The bit-error rate (BER) performance of different detectors versus SNR is plotted in Figure 2-4.

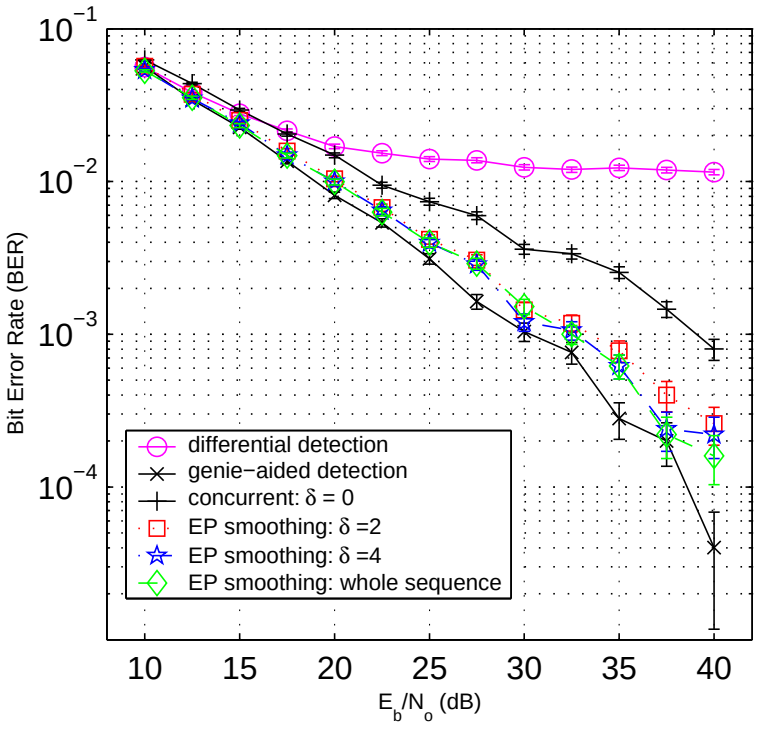


Figure 2-4: BER demodulation performance of the EP receiver with different smoothing parameters, the genie-aided detector, and the differential detector in a fading channel with complex Gaussian noises. The unit error bars, defined as $\sqrt{\frac{\text{BER}(1-\text{BER})}{T}}$, are also shown in the figure.

As shown in the figure, the window-based EP receiver clearly outperforms the concurrent detector and the differential detector. The new receiver does not have the error floor, while the differential detector does. With a delay $\delta = 2$, the performance of the window-based EP receiver is almost as good as that of the batch EP receiver, which performs smoothing over each entire data sequence, while the window-based EP receiver is more efficient and results in much shorter estimation delay than the batch EP receiver. Moreover, the performance of the window-based EP receiver with a delay $\delta = 3$ is close to the genie-aided detector.

Compared with the results obtained by Chen et al. (2000), the performance of the window-based EP receiver is comparable to that of a sequential Monte Carlo receiver. However, the EP receiver is much more efficient. Chen et al. (2000) use 50 samples in their sequential Monte Carlo receiver. With window length of 3, the window-based EP receiver is 13.33 times faster ($50 \cdot 2^3/(5 \cdot 2 \cdot 3) = 13.33$) than the sequential Monte Carlo receiver. Furthermore, the cost of the new receiver increases linearly with the window length, while the cost of the sequential Monte Carlo receiver explodes exponentially. For example, when the window length increases to 5, the new receiver becomes 32 times faster ($50 \cdot 2^5/(5 \cdot 2 \cdot 5) = 32$) than the sequential Monte Carlo receiver.

2.8 Discussion

This chapter has presented a window-based EP smoothing algorithm for online estimation. window-based EP smoothing achieves a trade-off between assumed density filtering and batch EP smoothing in terms of accuracy and efficiency.

For adaptive signal detection in fading channels, the window-based EP receiver significantly outperformed both the classical differential detector and the concurrent adaptive Bayesian receiver, which is based on ADF, under different signal-noise ratios. Moreover, the window-based EP receiver performs comparably to the batch EP receiver. However, the batch EP receiver uses the entire data sequence for smoothing and therefore leads to more estimation delay and larger computational cost. The performance of the window-based EP receiver is also close to the genie-aided detection, a performance upper bound. This performance similarity demonstrates the high quality of the window-based EP receiver. Compared to the sequential Monte Carlo receiver, the window-based EP receiver obtains the comparable estimation accuracy with less than one-tenth computational complexity. In

short, for the signal detection problem, window-based EP improves the estimation accuracy over ADF, enhances the efficiency over batch EP without sacrificing accuracy, and achieves comparable accuracy as the sequential Monte Carlo methods with much lower cost.

The window-based EP receiver can be used for many online estimation problems, such as object tracking in computer vision and iterative decoding in wireless digital communications. The EP receiver for wireless signal detection is just one example.

Chapter 3

Combining structured approximations with junction tree representation

3.1 Introduction

The previous section deals with dynamic graphical models, which are used to model sequential data. For data with more complicated structures, we need to consider more general graphical models, which may contain loops. For example, we may model a common sense database by a loopy graphical model, which compactly encodes the probabilistic statements in the database. Given such a probabilistic graphical model, we could answer a query to the common sense database by inferring the states of the graphical nodes associated with the query. Another example is sensor networks, where each sensor can be modeled by a node in a loopy graph. We can propagate probabilistic information in this loopy graph, in order to obtain consistent global knowledge of the environment from the distributed sensory information. Mathematically, a key problem is to estimate the marginal distributions of the variables indexed by the nodes in a loopy graph. For instance, consider the following toy problem. Given a grid in Figure 3-1 and its parameters, which specify the connection strength between different nodes, we want to estimate the marginal probability distributions $p(x_i), i = 1, \ldots, 16$.

[1]This chapter includes joint work with T.P. Minka (Minka & Qi, 2003).

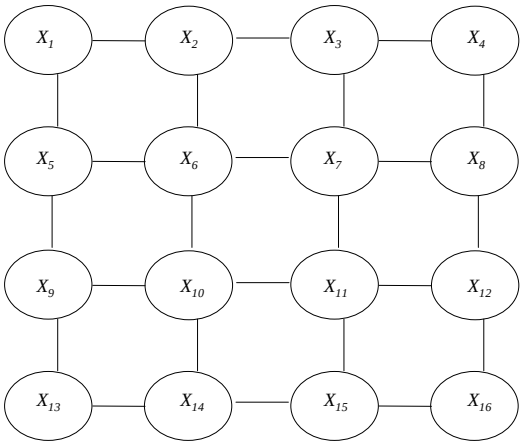


Figure 3-1: Example of loopy graph : grid

3.2 Approximate inference on loopy graphs

An important problem in deterministic approximate inference on loopy graphs is improving the performance of belief propagation (K. P. Murphy, 1999). Empirical studies have shown that belief propagation (BP) tends not to converge on graphs with strong positive and negative correlations (Welling & Teh, 2001). One approach that has been pursued in the literature is to force the convergence of BP in these cases, by appealing to a free-energy interpretation (Welling & Teh, 2001; Teh & Welling, 2001; Yuille, 2002). Unfortunately, this doesn't really solve the problem because it dramatically increases the computational cost and doesn't necessarily lead to good results on these graphs (Welling & Teh, 2001).

The expectation propagation (EP) framework (Minka, 2001) gives another interpretation of BP, as an algorithm which approximates multi-variable factors by single-variable factors ($f(x_1, x_2) \rightarrow \tilde{f}_1(x_1)\tilde{f}_2(x_2)$). This explanation suggests that it is BP's target approximation which is to blame, not the particular iterative scheme it uses. Factors which encode strong correlations obviously cannot be well approximated in this way. The connection between failure to converge and poor approximation holds true for EP algorithms in general, as shown by Minka (2001) and Heskes and Zoeter (2002).

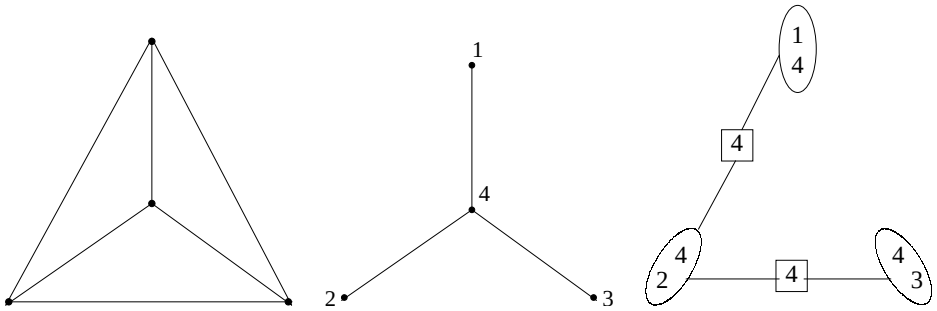
Working from the free-energy interpretation of BP, Yedidia et al. (2000) have suggested an extension involving the Kikuchi free-energy. The resulting algorithm resembles BP on a graph of node clusters, where again multi-variable factors are decomposed into independent parts ($f(x_1, x_2, x_3) \rightarrow \tilde{f}_1(x_1)\tilde{f}_{23}(x_2, x_3)$). This approach should work well for networks with

short-range correlations, such as grids. However, Kappen and Wiegerinck (2001) have shown that the Kikuchi method may give worse results than ordinary BP on densely connected graphs, and fixed-point Kikuchi iteration may fail to converge in cases where BP does converge.

On the other hand, Ghahramani and Jordan (1997) showed that tree structured approximations could improve the accuracy of variational bounds. Such bounds are tuned to minimize the 'exclusive' KL-divergence $KL(q||p)$, where q is the approximation. Frey et al. (2000) criticized this error measure and described an alternative method for minimizing the 'inclusive' divergence $KL(p||q)$. Their method, which sequentially projects graph potentials onto a tree structure, is closely related to expectation propagation. However, their method is not iterative and is therefore sensitive to the order in which the potentials are sequenced. Furthermore, Wainwright et al. (2001) and Wainwright et al. (2002) used tree structures on general graphical models. In the first paper, a "message-free" version of BP was derived, which used multiple tree structures to propagate evidence. The results it gives are nevertheless the same as BP. In the second paper, tree structures were used to obtain an upper bound on the normalizing constant of a Markov network. The trees produced by that method do not necessarily approximate the original distribution as a whole.

3.3 Combining tree-structured EP with junction tree algorithm

This section combines tree-structured EP approximation (Minka, 2001) with the junction tree algorithm (Madsen & Jensen, 1998). By using tree-structured approximation, this new approach is an extension of belief propagation, which has independent variable approximation. However, using a tree-structured approximation requires that every edge in a graph sends messages to all the other edges in a traditional EP algorithm. This requirement increases the computational cost dramatically. Therefore, we use the junction tree representation, which enables us to propagate messages in local regions to save cost.



p q junction tree of q

Figure 3-2: Approximating a complete graph p by a tree q. The junction tree of q is used to organize computations.

3.3.1 Using junction tree representation

Denote the original joint distribution by $p(\mathbf{x})$, written as a product of factors:

$$p(\mathbf{x}) = \prod_i f_i(\mathbf{x}) \tag{3.1}$$

For example, if $p(\mathbf{x})$ is a Bayesian network or Markov network, the factors are conditional probability distributions or potentials which each depend on a small subset of the variables in $\mathbf{x}$.

The target joint approximation $q(\mathbf{x})$ will have pairwise factors along a tree $\mathcal{T}$:

$$q(\mathbf{x}) = \frac{\prod_{(j,k)\in\mathcal{T}} q(x_j, x_k)}{\prod_{s\in\mathcal{S}} q(x_s)} \tag{3.2}$$

In this notation, $q(x_s)$ is the marginal distribution for variable x_s and $q(x_j, x_k)$ is the marginal distribution for the two variables x_j and x_k. These are going to be stored as multidimensional tables. The division is necessary to cancel overcounting in the numerator. A useful way to organize these divisions is to construct a *junction tree* connecting the cliques $(j,k) \in \mathcal{T}$ (Madsen & Jensen, 1998). This tree has a different structure than $\mathcal{T}$—the nodes in the junction tree represent cliques in $\mathcal{T}$, and the edges in the junction tree represent variables which are shared between cliques. These *separator* variables $\mathcal{S}$ in the junction tree are exactly the variables that go in the denominator of (3.2). Note that the same variable could be a separator more than once, so technically $\mathcal{S}$ is a multiset.

Figure 3-2 shows an example of how this all works. We want to approximate the distribution $p(\mathbf{x})$, which has a complete graph, by $q(\mathbf{x})$, whose graph is a spanning tree. The

marginal representation of q can be directly read off of the junction tree:

$$q(\mathbf{x}) = \frac{q(x_1, x_4)q(x_2, x_4)q(x_3, x_4)}{q(x_4)q(x_4)} \tag{3.3}$$

3.3.2 EP updates

The algorithm iteratively tunes $q(\mathbf{x})$ so that it matches $p(\mathbf{x})$ as closely as possible. Specifically, q tries to preserve the marginals and pairwise marginals of p:

$$q(x_j) \approx p(x_j) \tag{3.4}$$

$$q(x_j, x_k) \approx p(x_j, x_k) \qquad (j,k) \in \mathcal{T} \tag{3.5}$$

Expectation propagation is a general framework for approximating distributions of the form (3.1) by approximating the factors one by one. The final approximation q is then the product of the approximate factors. The functional form of the approximate factors is determined by considering the ratio of two different q's. In our case, this leads to approximations of the form

$$f_i(\mathbf{x}) \approx \tilde{f}_i(\mathbf{x}) = \frac{\prod_{(j,k)\in\mathcal{T}} \tilde{f}_i(x_j, x_k)}{\prod_{s\in\mathcal{S}} \tilde{f}_i(x_s)} \tag{3.6}$$

A product of such factors gives a distribution of the desired form (3.2). Note that $\tilde{f}_i(x_j, x_k)$ is not a proper marginal distribution, but just a non-negative function of two variables.

The algorithm starts by initializing the clique and separator potentials on the junction tree to 1. If a factor in p only depends on one variable, or variables which are adjacent in $\mathcal{T}$, then its approximation is trivial. It can be multiplied into the corresponding clique potential right away and removed from further consideration. The remaining factors in p, the *off-tree* factors, have their approximations $\tilde{f}_i$ initialized to 1.

To illustrate, consider the graph of Figure 3-2. Suppose all the potentials in p are pairwise, one for each edge. The edges $\{(1,4),(2,4),(3,4)\}$ are absorbed directly into q. The off-tree edges are $\{(1,2),(1,3),(2,3)\}$.

The algorithm then iteratively passes through the off-tree factors in p, performing the following three steps until all $\tilde{f}_i$ converge:

loop $i = 1, ..., M$:

(a) *Deletion.* Remove $\tilde{f}_i$ from q to get an 'old' approximation $q^{\setminus i}$:

$$q^{\setminus i}(x_j, x_k) = \frac{q(x_j, x_k)}{\tilde{f}_i(x_j, x_k)} \qquad (j,k) \in \mathcal{T} \tag{3.7}$$

$$q^{\setminus i}(x_s) = \frac{q(x_s)}{\tilde{f}_i(x_s)} \qquad s \in \mathcal{S} \tag{3.8}$$

(b) *Incorporate evidence.* Project the product of $f_i(\mathbf{x})$, considered as 'evidence', and $q^{\setminus i}$ into the junction tree by cutset conditioning (details in Section C, below). Propagate the evidence throughout the junction tree to obtain new clique marginals $q(x_j, x_k)$ and separators $q(x_s)$.

(c) *Update.* Reestimate $\tilde{f}_i$ by division:

$$\tilde{f}_i(x_j, x_k) = \frac{q(x_j, x_k)}{q^{\setminus i}(x_j, x_k)} \qquad (j,k) \in \mathcal{T} \tag{3.9}$$

$$\tilde{f}_i(x_s) = \frac{q(x_s)}{q^{\setminus i}(x_s)} \qquad s \in \mathcal{S} \tag{3.10}$$

3.3.3 Incorporating evidence by cutset conditioning

The purpose of the "incorporate evidence" step is to find a distribution q whose marginals match those of the product $f_i(\mathbf{x})q^{\setminus i}$. By definition, f_i depends on a set of variables which are not adjacent in $\mathcal{T}$, so the graph structure corresponding to $f_i(\mathbf{x})q^{\setminus i}(\mathbf{x})$ is not a tree, but has one or more loops. One approach is to apply a generic exact inference algorithm to $f_i(\mathbf{x})q^{\setminus i}(\mathbf{x})$ to obtain the desired marginals, e.g. construct a new junction tree in which $f_i(\mathbf{x})$ is a clique and propagate evidence in this tree. But this does not exploit the fact that we already have a junction tree for $q^{\setminus i}$ on which we can perform efficient inference.

Instead we use a more efficient approach—Pearl's cutset conditioning algorithm—to incorporate the evidence. Suppose $f_i(\mathbf{x})$ depends on a set of variables $\mathcal{V}$. The *domain* of $f_i(\mathbf{x})$ is the set of all possible assignments to $\mathcal{V}$. Find the clique $(j,k) \in \mathcal{T}$ which has the largest overlap with this domain—call this the *root clique*. Then enumerate the rest of the domain $V \setminus (x_j, x_k)$. For each possible assignment to these variables, enter it as evidence in q's junction tree and propagate to get marginals and an overall scale factor (which is the probability of that assignment). When the variables $\mathcal{V} \setminus (x_j, x_k)$ are fixed, entering evidence simply reduces to zeroing out conflicting entries in the junction tree, and multiplying the root clique (j,k) by $f_i(\mathbf{x})$. After propagating evidence multiple times, average the results

together according to their scale factors, to get the final marginals and separators of q.

Continuing the example of Figure 3-2, suppose we want to process edge $(1, 2)$, whose factor is $f_1(x_1, x_2)$. When added to q, this creates a loop. We cut the loop by conditioning on the variable with smallest range.Suppose x_1 is binary, so we condition on it. The other clique, $(2, 4)$, becomes the root. In one case, the evidence is $(x_1 = 0, f_1(0, x_2))$ and in the other it is $(x_1 = 1, f_1(1, x_2))$. Propagating evidence for both cases and averaging the results gives the new junction tree potentials.

Because it is an expectation-propagation algorithm, we know that a fixed point always exists, but we may not always find one. In these cases, the algorithm could be stabilized by a stepsize or double-loop iteration.

3.3.4 Local propagation

Another important optimization is proposed, by noting that evidence does not need to be propagated to the whole junction tree. In particular, it needs to be propagated only within the subtree that connects the nodes in $\mathcal{V}$. Evidence propagated to the rest of the tree will be exactly canceled by the separators, so even though the potentials may change, the ratios in (3.2) will not. For example, when we process edge $(1, 2)$ in Figure 3-2, there is no need to propagate evidence to clique $(3, 4)$, because when we divide $q(x_3, x_4)$ by the separator $q(x_4)$, we have $q(x_3|x_4)$, which is the same before and after the evidence.

Specifically, the algorithm incorporates each off-tree edge, i.e., evidence, as follows:

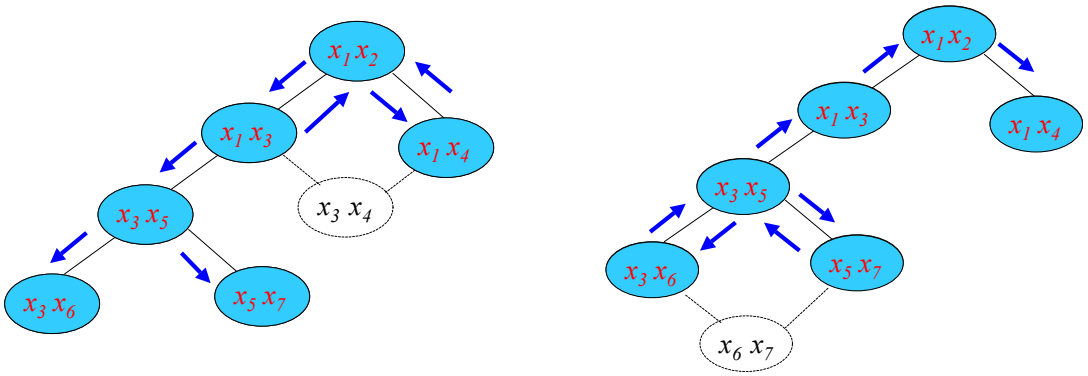
1. Find the subtree that is directly connected with the off-tree edge.

2. Incorporate the off-tree edge only into the subtree by the cutset conditioning method, instead of into the entire junction tree as described in Section 3.3.2. In other words, we update only clique marginals $q(x_j, x_k)$ and separators $q(x_s)$ in this subtree. Consequently, instead of computing the messages from this off-tree edge to the entire graph, as shown in (3.9) and (3.10), we only need to compute the messages in this subtree.

3. Find the (approximate) shortest path from the current subtree to the subtree connected with the off-tree edge to be processed next.

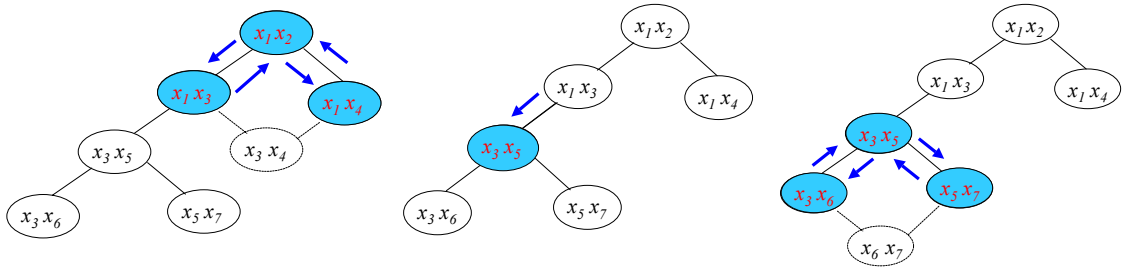
4. Pass evidence along this path using the junction tree algorithm.

5. Loop over all the off-tree edges.



(a) (b)

Figure 3-3: Global propagation for tree-structured EP approximation when incorporating the off-tree edge $\{3, 4\}$ or $\{6, 7\}$ in (a) and (b), respectively. The messages, indicated by directed bold line segments, are sent to the whole tree from the edge $\{3, 4\}$ or $\{6, 7\}$. In this junction tree representation, we ignore the separators without causing confusion.



(a) (b) (c)

Figure 3-4: Local propagation for tree-structured EP approximation: (a) incorporating the off-tree edge $\{3, 4\}$; (b) propagating information between subtrees; (c) incorporating the off-tree edge $\{6, 7\}$. The messages are sent only to the subtree that is directly connected to the off-tree edge being processed. This local propagation scheme reduces computational cost and saves memory.

Figures 3-3 and 3-4 illustrate the difference between global and local propagations. To incorporate edge $\{3, 4\}$, the global propagation scheme sends messages to the entire tree, while the local propagation scheme sends messages only to the cliques $\{1, 4\}$, $\{1, 2\}$, and $\{1, 3\}$ in the subtree. The local propagation scheme then propagates evidence from this subtree to the subtree connected to the next off-tree edge $\{6, 7\}$, by performing junction-tree updates from clique $\{1, 3\}$ to $\{3, 5\}$. Similarly, we can incorporate the off-tree edge $\{6, 7\}$. Even in this simple graph, local propagation runs roughly twice as fast as global propagation, and uses about half the memory to store messages. On larger graphs used in

the experiment section, the improvement is also greater.

As demonstrated in this example, we can interpret the algorithm based on local propagation as a combination of the junction tree algorithm with EP: on the one hand, the algorithm performs traditional junction-tree propagation, and on the other hand, it projects off-tree edges into subtrees by the cutset conditioning method. Clearly, when we do not have any off-tree edge, this algorithm reduces exactly to the junction tree algorithm.

3.4 Choosing the tree structure

This section describes a simple method to choose the tree structure. It leaves open the problem of finding the 'optimal' approximation structure; instead, it presents a simple rule which works reasonably well in practice.

Intuitively, we want edges between the variables which are the most correlated. The approach is based on Chow and Liu (1968): estimate the mutual information between adjacent nodes in p's graph, call this the 'weight' of the edge between them, and then find the spanning tree with maximal total weight. The mutual information between nodes requires an estimate of their joint distribution. In our implementation, this is obtained from the product of factors involving only these two nodes, i.e. the single-node potentials times the edge between them. While crude, it does capture the amount of correlation provided by the edge, and thus whether we should have it in the approximation.

3.5 Numerical results

This section compares the new algorithm with different alternatives, including belief propagation, generalized belief propagation, and structured and naive mean fields, for inference on loopy graphs.

3.5.1 The four-node network

This section illustrates the algorithm on a concrete problem, comparing it to other methods for approximate inference. The network and approximation will be the ones pictured in Figure 3-2, with all nodes binary. The potentials were chosen randomly and can be obtained from the authors' website.

Method	FLOPS	$E[x_1]$	$E[x_2]$	$E[x_3]$	$E[x_4]$	Error
Exact	200	0.474	0.468	0.482	0.536	0
TreeEP	800	0.467	0.459	0.477	0.535	0.008
GBP	2200	0.467	0.459	0.477	0.535	0.008
TreeVB	11700	0.460	0.460	0.476	0.540	0.014
BP	500	0.499	0.499	0.5	0.501	0.035
MF	11500	0.000	0.000	0.094	0.946	0.474

Table 3.1: Node means estimated by various methods (TreeEP = the proposed method, BP = loopy belief propagation, GBP = generalized belief propagation on triangles, MF = mean-field, TreeVB = variational tree). FLOPS are rounded to the nearest hundred.

Five approximate inference methods were compared. The proposed method (TreeEP) used the tree structure specified in Figure 3-2. Mean-field (MF) fit a variational bound with independent variables, and TreeVB fit a tree-structured variational bound, with the same structure as TreeEP. TreeVB was implemented using the general method described by Wiegerinck (2000), with the same junction tree optimizations as in TreeEP.

Generalized belief propagation (GBP) was implemented using the parent-child algorithm of Yedidia et al. (2002) (with special attention to the damping described in section 8). We also used GBP to perform ordinary loopy belief propagation (BP). Our implementation tries to be efficient in terms of FLOPS, but we do not know if it is the fastest possible. GBP and BP were first run using stepsize 0.5, and if didn't converge, halved it and started over. The time for these 'trial runs' was not counted.

The algorithms were all implemented in Matlab using Kevin Murphy's BNT toolbox (Murphy, 2001). Computational cost was measured by the number of floating-point operations (FLOPS). Because the algorithms are iterative and can be stopped at any time to get a result, we used a "5% rule" to determine FLOPS. The algorithm was run for a large number of iterations, and the error at each iteration was computed. At each iteration, we then get an error *bound*, which is the maximum error from that iteration onwards. The first iteration whose error bound is within 5% of the final error is chosen for the official FLOP count. (The official error is still the final error.)

The results are shown in Table 3.1. TreeEP is more accurate than BP, with less cost than TreeVB and GBP. GBP was run with clusters $\{(1,2,4),(1,3,4),(2,3,4)\}$. This gives the same result as TreeEP, because these clusters are exactly the off-tree loops.

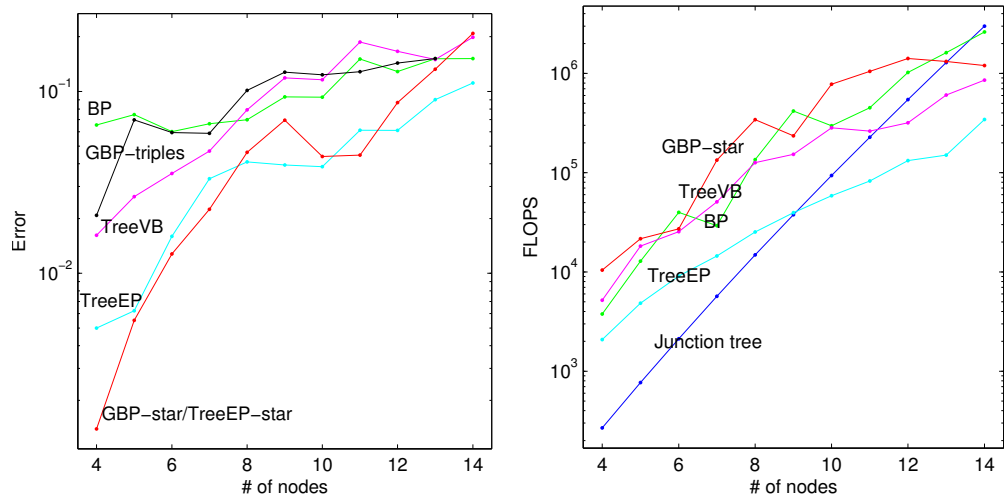


Figure 3-5: (a) Error in the estimated means for complete graphs with randomly chosen potentials. Each point is an average over 10 potentials. (b) Average FLOPS for the results in (a).

3.5.2 Complete graphs

The next experiment tests the algorithms on complete graphs of varying size. The graphs have random single-node and pairwise potentials, of the form $f_i(x_j) = [\exp(\theta_j)\exp(-\theta_j)]$ and $f_i(x_j, x_k) = \begin{bmatrix} \exp(w_{jk}) & \exp(-w_{jk}) \\ \exp(-w_{jk}) & \exp(w_{jk}) \end{bmatrix}$. The "external fields" θ_j were drawn independently from a Gaussian with mean 0 and standard deviation 1. The "couplings" w_{jk} were drawn independently from a Gaussian with mean 0 and standard deviation $3/\sqrt{n-1}$, where n is the number of nodes. Each node has $n-1$ neighbors, so this tries to keep the overall coupling level constant.

Figure 3-5(a) shows the approximation error as n increases. For each n, 10 different potentials were drawn, giving 110 networks in all. For each one, the maximum absolute difference between the estimated means and exact means was computed. These errors are averaged over potentials and shown separately for each graph size. TreeEP and TreeVB always used the same structure, picked according to section 3.4. TreeEP outperforms BP consistently, but TreeVB does not.

For this type of graph, we found that GBP works well with clusters in a 'star' pattern, i.e. the clusters are $\{(1,2,3),(1,3,4),(1,4,5),...,(1,n,2)\}$. Node '1' is the center of the star, and was chosen to be the node with highest average coupling to its neighbors. As shown in Figure 3-5(a), this works much better than using all triples of nodes, as done by Kappen

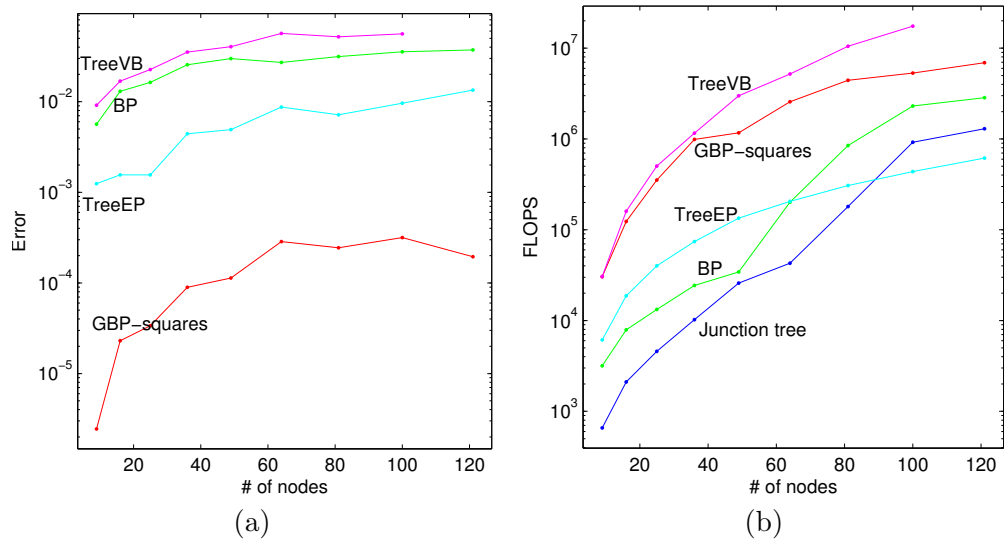


Figure 3-6: (a) Error in the estimated means for grid graphs with randomly chosen potentials. Each point is an average over 10 potentials. (b) Average FLOPS for the results in (a).

and Wiegerinck (2001). Note that if TreeEP is given a similar 'star' structure, the results are the same as GBP. This is because the GBP clusters coincide with the off-tree loops. In general, if the off-tree loops are triangles, then GBP on those triangles will give identical results.

Figure 3-5(b) shows the cost as n increases. TreeEP and TreeVB scale the best, with TreeEP being the fastest method on large graphs.

3.5.3 Grids

The next experiment tests the algorithms on square grids of varying size. The external fields θ_j were drawn as before, and the couplings w_{jk} had standard deviation 1. The GBP clusters were overlapping squares, as in Yedidia et al. (2000).

Figure 3-6(a) shows the approximation error as n increases, with results averaged over 10 trials as in the previous section. TreeVB performs consistently worse than BP, even though it is using the same tree structures as TreeEP. The plot also shows that these structures, being automatically chosen, are not as good as the hand-crafted clusters used by GBP. We have hand-crafted tree structures that perform just as well on grids, but for simplicity we do not include these results.

Figure 3-6(b) shows that TreeEP is the fastest on large grids, even faster than BP, because BP must use increasingly smaller stepsizes. GBP is more than a factor of ten

slower.

3.6 Conclusions

Combining tree-structured EP approximation with the junction tree representation enables TreeEP to propagate information in a subtree, and to maintain only local consistency, which greatly reduces the computation and the memory cost. As a result, TreeEP allows a smooth tradeoff between cost and accuracy in approximate inference. It improves on BP for a modest increase in cost. In particular, when ordinary BP doesn't converge, TreeEP is an attractive alternative to damping or double-loop iteration. TreeEP performs better than the corresponding variational bounds, because it minimizes the inclusive KL-divergence. We found that TreeEP was equivalent to GBP in some cases, which deserves further study. We hope that these results encourage more investigation into approximation structure for inference algorithms, such as finding the "optimal" structure for a given problem.

Chapter 4

Predictive automatic relevance determination

4.1 Introduction

This chapter[1] moves the focus from generative models in Chapters 2 and 3 to conditional classification models, and presents a new approach for Bayesian feature selection, which enhances EP classification in the presence of irrelevant features. This new approach can also used for sparse kernel learning, which sparsifies EP classifiers for fast test performance.

In many real-world classification and regression problems the input consists of a large number of features or variables, only some of which are relevant. Inferring which inputs are relevant is an important problem. It has received a great deal of attention in machine learning and statistics over the last few decades (Guyon & Elisseeff, 2003).

This paper focuses on Bayesian approaches to determining the relevance of input features. One of the most successful methods is called *automatic relevance determination* (ARD) (MacKay, 1992; Neal, 1996). This is a hierarchical Bayesian approach where there are hyperparameters which explicitly represent the relevance of different input features. These relevance hyperparameters determine the range of variation for the parameters relating to a particular input, usually by modeling the width of a zero-mean Gaussian prior on those parameters. If the width of that Gaussian is zero, then those parameters are constrained to be zero, and the corresponding input cannot have any effect on the predictions,

[1]This chapter includes the joint work with T.P. Minka, R. W. Picard, and Z. Ghahramani (Qi et al., 2004).

therefore making it irrelevant. ARD optimizes these hyperparameters to discover which inputs are relevant. [2]

Automatic relevance determination optimizes the *model evidence*, also known as the *marginal likelihood*, which is the classic criterion used for Bayesian model selection. In this paper we show that while this is often effective, in cases where there are a very large number of input features it can lead to overfitting. We instead propose a different approach, called *predictive ARD*, which is based on the estimated predictive performance. We show that this estimated predictive performance can be computed efficiently as a side effect of the expectation propagation algorithm for approximate inference and that it performs better that the evidence-based ARD on a variety of classification problems.

Although the framework we present can be applied to many Bayesian classification and regression models, we focus our presentation and experiments on classification problems in the presence of irrelevant features as well as in sparse Bayesian learning for kernel methods.

Compared to the traditional ARD classification, this paper presents three specific enhancements: (1) an approximation of the integrals via Expectation Propagation, instead of Laplace's method or Monte Carlo; (2) an ARD procedure which minimizes an estimate of the predictive leave-one-out generalization error (obtained directly from EP); (3) a fast sequential update for the hyperparameters based on Faul and Tipping (2002)'s recent work. These enhancements improve classification performance.

The rest of this chapter is organized as follows. Section 2 reviews the ARD approach to classification and its properties. Section 3 presents predictive ARD by EP, followed by experiments and discussions in section 4.

4.2 Automatic relevance determination

A linear classifier classifies a point $\mathbf{x}$ according to $t = \text{sign}(\mathbf{w}^{\mathrm{T}}\mathbf{x})$ for some parameter vector $\mathbf{w}$ (the two classes are $t = \pm 1$). Given a training set $D = \{(\mathbf{x}_1, t_1), ..., (\mathbf{x}_N, t_N)\}$, the likelihood for $\mathbf{w}$ can be written as

$$p(\mathbf{t}|\mathbf{w}, X) = \prod_i p(t_i|\mathbf{x}_i, \mathbf{w}) = \prod_i \Psi(t_i \mathbf{w}^{\mathrm{T}} \boldsymbol{\phi}(\mathbf{x}_i)) \tag{4.1}$$

[2] From a strictly Bayesian point of view, hard decisions about whether an input feature should be selected or not are not warranted unless there is a loss function which explicitly associates a cost to the number of features. However, in practice it is often desirable to have an easily interpretable model with a sparse subset of relevant features, and ARD methods can achieve this while closely approximating the full Bayesian average which does no feature selection.

where $\mathbf{t} = \{t_i\}_{i=1}^N$, $X = \{\mathbf{x}_i\}_{i=1}^N$, $\Psi(\cdot)$ is the cumulative distribution function for a Gaussian. One can also use the step function or logistic function as $\Psi(\cdot)$. The basis function $\boldsymbol{\phi}^T(\mathbf{x}_i)$ allows the classification boundary to be nonlinear in the original features. This is the same likelihood used in logistic regression and in Gaussian process classifiers. Given a new input $\mathbf{x}_{N+1}$, we approximate the predictive distribution:

$$p(t_{N+1}|\mathbf{x}_{N+1}, \mathbf{t}) = \int p(t_{N+1}|\mathbf{x}_{N+1}, \mathbf{w}) p(\mathbf{w}|\mathbf{t}) d\mathbf{w} \tag{4.2}$$

$$\approx p(t_{N+1}|\mathbf{x}_{N+1}, \langle \mathbf{w} \rangle) \tag{4.3}$$

where $\langle \mathbf{w} \rangle$ denotes the posterior mean of the weights, called the Bayes Point (Herbrich et al., 1999).

The basic idea in ARD is to give the feature weights independent Gaussian priors:

$$p(\mathbf{w}|\boldsymbol{\alpha}) = \prod_i \mathcal{N}(w_i|0, \alpha_i^{-1}),$$

where $\boldsymbol{\alpha} = \{\alpha_i\}$ is a hyperparameter vector that controls how far away from zero each weight is allowed to go. The hyperparameters $\boldsymbol{\alpha}$ are trained from the data by maximizing the Bayesian 'evidence' $p(\mathbf{t}|\boldsymbol{\alpha})$, which can be done using a fixed point algorithm or an EM algorithm treating $\mathbf{w}$ as a hidden variable (MacKay, 1992). The outcome of this optimization is that many elements of $\boldsymbol{\alpha}$ go to infinity such that $\mathbf{w}$ would have only a few nonzero weights w_j. This naturally prunes irrelevant features in the data. Later we will discuss why ARD favors sparse models (section 4.2.2). The Bayesian ARD model is illustrated in Figure 4-1.

4.2.1 ARD-Laplace

Both the fixed point and EM algorithms require the posterior moments of $\mathbf{w}$. These moments have been approximated by second-order expansion, i.e. Laplace's method (MacKay, 1992), or approximated by Monte Carlo (Neal, 1996). ARD with Laplace's method (**ARD-Laplace**) was used in the Relevance Vector Machine (RVM) (Tipping, 2000). The RVM is a linear classifier using basis functions $\boldsymbol{\phi}(\mathbf{x}) = [k(\mathbf{x}, \mathbf{x}_1), k(\mathbf{x}, \mathbf{x}_2), \cdots, k(\mathbf{x}, \mathbf{x}_N)]$. Specifically, Laplace's method approximates the evidence by a Gaussian distribution around the

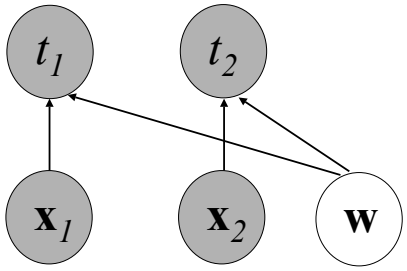


Figure 4-1: Graphical model for Bayesian conditional classification. The shaded nodes represent observed variables, i.e., two input data points, and the unshaded node represents latent variables, i.e., the model parameters $\mathbf{w}$. Using a Bayesian approach, we want to estimate the posterior distribution of the parameters $\mathbf{w}$ in the graph. Automatic relevance determination maximizes the evidence over hyperparameters, which control the variance of the prior distribution of $\mathbf{w}$, to prune irrelevant features.

maximum a posteriori (MP) value of $\mathbf{w}$, $\mathbf{w}_{MP}$, as follows:

$$\begin{aligned}
\Sigma = -\mathbf{H}^{-1} &= -\frac{d^2 \log p(\mathbf{w}, \mathbf{t}|\boldsymbol{\alpha})}{d\mathbf{w} d\mathbf{w}^T}\Big|^{-1}_{\mathbf{w}=\mathbf{w}_{MP}} \\
p(\mathbf{w}, \mathbf{t}|\boldsymbol{\alpha}) &\approx p(\mathbf{t}, \mathbf{w}_{MP}) \exp\big(-\frac{1}{2}(\mathbf{w}-\mathbf{w}_{MP})^T \Sigma^{-1} (\mathbf{w}-\mathbf{w}_{MP})\big) \\
p(\mathbf{t}|\boldsymbol{\alpha}) &= \int p(\mathbf{w}, \mathbf{t}|\boldsymbol{\alpha}) d\mathbf{w} \approx p(\mathbf{t}, \mathbf{w}_{MP}) |2\pi\Sigma|^{1/2} \\
p(\mathbf{w}|\mathbf{t}, \boldsymbol{\alpha}) &= \frac{p(\mathbf{w}, \mathbf{t}|\boldsymbol{\alpha})}{p(\mathbf{t}|\boldsymbol{\alpha})} \approx \mathcal{N}(\mathbf{w}|\mathbf{w}_{MP}, \Sigma)
\end{aligned}$$

If we use a logistic model for $\Psi(\cdot)$, then the Hessian matrix $\mathbf{H}$ has the following form: $H = -(\Phi B \Phi^{\mathrm{T}} + A)$, where $\Phi = \big(\phi(\mathbf{x}_i), \ldots, \phi(\mathbf{x}_N)\big)$ is a d by N matrix, $A = \mathrm{diag}(\boldsymbol{\alpha})$, and B is a diagonal matrix with $B_{ii} = \Psi\big(\mathbf{w}_{MP}^{\mathrm{T}} \mathbf{x}_i\big)\big(1 - \Psi(\mathbf{w}_{MP}^{\mathrm{T}} \mathbf{x}_i)\big)$.

Laplace's method is a simple and powerful approach for approximating a posterior distribution. But it does not really try to approximate the posterior mean; instead it simply approximates the posterior mean by the posterior mode. The quality of the approximation for the posterior mean can be improved by using EP as shown by Minka (2001).

4.2.2 Overfitting of ARD

Overfitting can be caused not only by over-complicated classifiers, but also by just picking one from many simple classifiers that can correctly classify the data. Consider the example

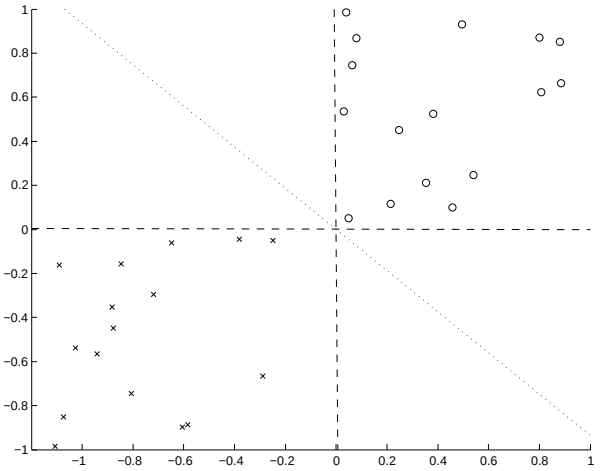


Figure 4-2: Illustration of overfitting of ARD.

plotted in Figure 4-2. The data labelled with 'x' and 'o' are in class 1 and 2 respectively. Every line through the origin with negative slope separates the data. If you apply the regular Bayes Point linear classifier, you get a classifier of angle 135° (shown); if you apply ARD to maximize evidence, you end up with a horizontal line which is sparse, because it ignores one input feature, but seemingly very dangerous. Both horizontal and vertical sparse classifiers have larger evidence values than the one of 135°, though both of them are intuitively more dangerous. Having effectively pruned out one of the two parameter dimensions (by taking one of the αs to infinity), the evidence for the horizontal and vertical classifiers involves computing an integral over only one remaining dimension. However, the evidence for the classifier which retains both input dimensions is an integral over two parameter dimensions. In general, a more complex model will have lower evidence than a simpler model if they can both classify the data equally well. Thus ARD using evidence maximization chooses the "simpler" model, which in this case is more dangerous because it uses only one relevant dimension.

ARD is a Type II maximum likelihood method and thus subject to overfitting as the example above illustrates. However, it is important to point out that the overfitting resulting from fitting the relevance hyperparameters $\boldsymbol{\alpha}$ is not the same kind of overfitting as one gets from fitting the parameters $\mathbf{w}$ as in classical maximum likelihood methods. By optimizing $\mathbf{w}$ one can directly fit noise in the data. However, optimizing $\boldsymbol{\alpha}$ only corresponds to making decisions about which variables are irrelevant, since $\mathbf{w}$ is integrated out. In the simple case where α can take only two values, very large or small, and the input is d-dimensional,

choosing $\boldsymbol{\alpha}$ corresponds to model selection by picking one out of 2^d subsets of input features. This is only d bits of information in the training data that can be overfit, far fewer than what can be achieved by precisely tuning a single real-valued parameter w. However, when d is large, as in the practical examples in our experimental section, we find that even this overfitting can cause problems. This motivates our proposed predictive measure for ARD based on estimating leave-one-out performance.

4.2.3 Computational issues

To compute the posterior moments of $\mathbf{w}$ required by ARD-Laplace, we need to invert the Hessian matrix $\mathbf{H}$ for each update. This takes $O(d^3)$ time, which is quite expensive when the dimension d is large.

4.3 Predictive-ARD-EP

In this section, we improve ARD-Laplace in three ways: replacing Laplace's method by more accurate EP, estimating the predictive performance based on leave-one-out estimate without actually carrying out the expensive cross-validation, and incorporating a fast sequential optimization method into ARD-EP.

4.3.1 EP for probit model

The algorithm described in this section is a variant of the one in (Minka, 2001), and we refer the reader to that paper for a detailed derivation. Briefly, Expectation Propagation (EP) exploits the fact that the likelihood is a product of simple terms. If we approximate each of these terms well, we can get a good approximation to the posterior. Expectation Propagation chooses each approximation such that the posterior using the term exactly and the posterior using the term approximately are close in KL-divergence. This gives a system of coupled equations for the approximations which are iterated to reach a fixed-point.

Denote the exact terms by $g_i(\mathbf{w})$ and the approximate terms by $\tilde{g}_i(\mathbf{w})$:

$$\begin{aligned} p(\mathbf{w}|\mathbf{t}, \boldsymbol{\alpha}) &\propto p(\mathbf{w}|\boldsymbol{\alpha}) \prod_i p(t_i|\mathbf{w}) \\ &= p(\mathbf{w}|\boldsymbol{\alpha}) \prod_i g_i(\mathbf{w}) \approx p(\mathbf{w}|\boldsymbol{\alpha}) \prod_i \tilde{g}_i(\mathbf{w}) \end{aligned}$$

The approximate terms are chosen to be Gaussian, parameterized by (m_i, v_i, s_i): $\tilde{g}_i = s_i \exp(-\frac{1}{2v_i}(t_i\boldsymbol{\phi}_i^T\mathbf{w} - m_i)^2)$. This makes the approximate posterior distribution also Gaussian: $p(\mathbf{w}|\mathbf{t}, \boldsymbol{\alpha}) \approx q(\mathbf{w}) = \mathcal{N}(\mathbf{m}_w, \mathbf{V}_w)$.

To find the best term approximations, proceed as follows: (to save notation, $t_i\boldsymbol{\phi}_i$ is written as $\boldsymbol{\phi}_i$)

1. Initialization Step: Set $\tilde{g}_i = 1$: $v_i = \infty$, $m_i = 0$, and $s_i = 1$. Also, set $q(\mathbf{w}) = p(\mathbf{w}|\boldsymbol{\alpha})$.

2. Loop until all (m_i, v_i, s_i) converge:

   Loop $i = 1, \ldots, N$:

   (a) Remove the approximation $\tilde{g}_i$ from $q(\mathbf{w})$ to get the 'leave-one-out' posterior $q^{\setminus i}(\mathbf{w})$, which is also Gaussian: $\mathcal{N}(\mathbf{m}_w^{\setminus i}, \mathbf{V}_w^{\setminus i})$. From $q^{\setminus i}(\mathbf{w}) \propto q(\mathbf{w})/\tilde{g}_i$, this implies

$$\mathbf{V}_w^{\setminus i} = \mathbf{V}_w + \frac{(\mathbf{V}_w\boldsymbol{\phi}_i)(\mathbf{V}_w\boldsymbol{\phi}_i)^T}{v_i - \boldsymbol{\phi}_i^T\mathbf{V}_w\boldsymbol{\phi}_i} \tag{4.4}$$

$$\mathbf{m}_w^{\setminus i} = \mathbf{m}_w + (\mathbf{V}_w^{\setminus i}\boldsymbol{\phi}_i)v_i^{-1}(\boldsymbol{\phi}_i^T\mathbf{m}_w - m_i) \tag{4.5}$$

   (b) Putting the posterior without i together with term i gives $\hat{p}(\mathbf{w}) \propto g_i(\mathbf{w})q^{\setminus i}(\mathbf{w})$. Choose $q(\mathbf{w})$ to minimize $KL(\hat{p}(\mathbf{w}) \,||\, q(\mathbf{w}))$. Let Z_i be the normalizing factor.

$$\begin{aligned}
\mathbf{m}_w &= \mathbf{m}_w^{\setminus i} + \mathbf{V}_w^{\setminus i}\rho_i\boldsymbol{\phi}_i \\
\mathbf{V}_w &= \mathbf{V}_w^{\setminus i} - (\mathbf{V}_w^{\setminus i}\boldsymbol{\phi}_i)\Big(\frac{\rho_i(\boldsymbol{\phi}_i^T\mathbf{m}_w + \rho_i)}{\boldsymbol{\phi}_i^T\mathbf{V}_w^{\setminus i}\boldsymbol{\phi}_i + 1}\Big)(\mathbf{V}_w^{\setminus i}\boldsymbol{\phi}_i)^T \\
Z_i &= \int_{\mathbf{w}} g_i(\mathbf{w})q^{\setminus i}(\mathbf{w})\mathrm{d}\mathbf{w} = \Psi(z_i)
\end{aligned} \tag{4.6}$$

   where

$$z_i = \frac{(\mathbf{m}_w^{\setminus i})^T\boldsymbol{\phi}_i}{\sqrt{\boldsymbol{\phi}_i^T\mathbf{V}_w^{\setminus i}\boldsymbol{\phi}_i + 1}} \qquad \rho_i = \frac{1}{\sqrt{\boldsymbol{\phi}_i^T\mathbf{V}_w^{\setminus i}\boldsymbol{\phi}_i + 1}}\frac{\mathcal{N}(z_i; 0, 1)}{\Psi(z_i)} \tag{4.7}$$

   (c) From $\tilde{g}_i = Z_i\frac{q(\mathbf{w})}{q^{\setminus i}(\mathbf{w})}$, update the term approximation:

$$v_i = \boldsymbol{\phi}_i^T\mathbf{V}_w^{\setminus i}\boldsymbol{\phi}_i\Big(\frac{1}{\rho_i(\boldsymbol{\phi}_i^T\mathbf{m}_w + \rho_i)} - 1\Big) + \frac{1}{\rho_i(\boldsymbol{\phi}_i^T\mathbf{m}_w + \rho_i)} \tag{4.8}$$

$$m_i = \boldsymbol{\phi}_i^T \mathbf{m}_w^{\setminus i} + (v_i + \boldsymbol{\phi}_i^T \mathbf{V}_w^{\setminus i} \boldsymbol{\phi}_i)\rho_i \tag{4.9}$$

$$s_i = \Psi(z_i)\sqrt{1 + v_i^{-1} \boldsymbol{\phi}_i^T \mathbf{V}_w^{\setminus i} \boldsymbol{\phi}_i} \exp\big(\frac{1}{2} \frac{\boldsymbol{\phi}_i^T \mathbf{V}_w^{\setminus i} \boldsymbol{\phi}_i + 1}{\boldsymbol{\phi}_i^T \mathbf{m}_w + \rho_i} \rho_i\big) \tag{4.10}$$

3. Finally, compute the normalizing constant and the evidence:

$$B = (\mathbf{m}_w)^T \mathbf{V}_w^{-1} \mathbf{m}_w - \sum_i \frac{m_i^2}{v_i}$$

$$p(D|\boldsymbol{\alpha}) \approx \int \prod_i p(\mathbf{w}|\boldsymbol{\alpha}) \tilde{g}_i(\mathbf{w}) \mathrm{d}\mathbf{w} = \frac{|\mathbf{V}_w|^{1/2}}{(\prod_j \alpha_j)^{1/2}} \exp(B/2) \prod_i s_i \tag{4.11}$$

The time complexity of this algorithm is $O(d^2)$ for processing each term, and therefore $O(Nd^2)$ per iteration.

4.3.2 Estimate of predictive performance

A nice property of EP is that it can easily offer an estimate of leave-one-out error without any extra computation. At each iteration, EP computes in (4.4) and (4.5) the parameters of the approximate leave-one-out posterior $q^{\setminus i}(\mathbf{w})$ that does not depend on the i^{th} data point. So we can use the mean $\mathbf{m}_w^{\setminus i}$ to approximate a classifier trained on the other $(N-1)$ data points. Thus an estimate of leave-one-out error can be computed as

$$\epsilon_{loo} = \frac{1}{N} \sum_{i=1}^{N} \Theta(-t_i (\mathbf{m}_w^{\setminus i})^T \boldsymbol{\phi}(\mathbf{x}_i)) \tag{4.12}$$

where $\Theta(\cdot)$ is a step function. An equivalent estimate was given by Opper and Winther (2000) using the TAP method for training Gaussian processes, which is equivalent to EP (Minka, 2001).

Furthermore, we can provide an estimate of leave-one-out error probability. Since Z_i in (4.6) is the posterior probability of the i^{th} data label, we propose the following estimator:

$$\epsilon_{pred} = \frac{1}{N} \sum_{i=1}^{N} (1 - Z_i) = \frac{1}{N} \sum_{i=1}^{N} \Psi(-z_i) \tag{4.13}$$

where Z_i and z_i is defined in (4.6) and (4.7). In (4.7), $\boldsymbol{\phi}_i$ is the product of t_i and $\phi(\mathbf{x}_i)$. By contrast, Opper and Winther estimate the error probability by $\frac{1}{N} \sum_{i=1}^{N} \Psi(-|z_i|)$, which ignores the information of the label t_i. Notice that ϵ_{pred} utilizes the variance of $\mathbf{w}$ given the

data, not just the mean. However, it assumes that the posterior for $\mathbf{w}$ has Gaussian tails. In reality, the tails are lighter so we compensate by pre-scaling z_i by 50, a number tuned by simulations.

4.3.3 Fast optimization of evidence

This section combines EP with a fast sequential optimization method (Faul & Tipping, 2002) to efficiently update the hyperparameters $\boldsymbol{\alpha}$.

As mentioned before, EP approximates each classification likelihood term $g_i(\mathbf{w}) = \Psi(\mathbf{w}^{\mathrm{T}}\phi_i)$ by $\tilde{g}_i = s_i \exp(-\frac{1}{2v_i}(\boldsymbol{\phi}_i^T\mathbf{w} - m_i)^2)$. Here ϕ_i is short hand for $t_i\phi(\mathbf{x}_i)$ as in the previous section. Notice that $\tilde{g}_i$ has the same form as a regression likelihood term in a regression problem. Therefore, EP actually maps a classification problem into a regression problem where (m_i, v_i) defines the virtual observation data point with mean m_i and variance v_i. Based on this interpretation, it is easy to see that for the approximate posterior $q(\mathbf{w})$, we have

$$\mathbf{V}_w = \left(\mathbf{A} + \Phi\Lambda^{-1}\Phi^{\mathrm{T}}\right)^{-1} \qquad \mathbf{m}_w = \mathbf{V}_w\Phi\Lambda^{-1}\mathbf{m}_o \tag{4.14}$$

where we define

$$\mathbf{m}_o = (m_1, \ldots, m_N)^{\mathrm{T}} \qquad \mathbf{A} = \mathrm{diag}(\boldsymbol{\alpha})$$
$$\mathbf{v}_o = (v_1, \ldots, v_N)^{\mathrm{T}} \qquad \Lambda = \mathrm{diag}(\mathbf{v}_o)$$

and $\Phi = (\phi_i, \ldots, \phi_N)$ is a d by N matrix.

To have a sequential update on α_j, we can explicitly decompose $p(D|\boldsymbol{\alpha})$ into two parts, one part denoted by $p(D|\boldsymbol{\alpha}_{\setminus j})$, that does not depend on α_j and another that does, i.e.,

$$p(D|\boldsymbol{\alpha}) = p(D|\boldsymbol{\alpha}_{\setminus j}) + \frac{1}{2}\big(\log\alpha_j - \log(\alpha_j + r_j) + \frac{u_j^2}{\alpha_j + r_j}\big)$$

where $r_j = \phi_j\mathbf{C}_{\setminus j}^{-1}\phi_j^{\mathrm{T}}$, $u_j = \phi_j\mathbf{C}_{\setminus j}^{-1}\mathbf{m}_o$, and $\mathbf{C}_{\setminus j} = \Lambda^{-1} + \sum_{m\neq j}\phi_m^{\mathrm{T}}\phi_m$. Here ϕ_j and ϕ_m are j^{th} and m^{th} rows of the data matrix Φ respectively.

Using the above equation, Faul and Tipping (2002) show $p(D|\boldsymbol{\alpha})$ has a maximum with

respect to α_j:

$$\alpha_j = \frac{r_j^2}{u_j^2 - r_j}, \qquad \text{if} \quad \eta_j > 0 \tag{4.15}$$

$$\alpha_j = \infty, \qquad \text{if} \quad \eta_j \leq 0 \tag{4.16}$$

where $\eta_j = u_j^2 - r_j$. Thus, in order to maximize the evidence, we introduce the j^{th} feature when $\alpha_j = \infty$ and $\eta_j > 0$, exclude the j^{th} feature when $\alpha_j < \infty$ and $\eta_j \leq 0$, and reestimate α_j according to (4.15) when $\alpha_j < \infty$ and $\eta_j > 0$. To further save computation, we can exploit the following relations:

$$r_j = \frac{\alpha_j R_j}{\alpha_j - R_j}, \quad u_j = \frac{\alpha_j U_j}{\alpha_j - R_j} \tag{4.17}$$

where $R_j = \phi_j \Lambda^{-1} \phi_j^{\mathrm{T}} - \phi_j \Lambda^{-1} \hat{\Phi}^{\mathrm{T}} \hat{\mathbf{m}}_w$ and $U_j = \phi_j \Lambda^{-1} \mathbf{m}_o - \phi_j \Lambda^{-1} \hat{\Phi}^{\mathrm{T}} \hat{\mathbf{V}}_w \hat{\Phi} (\phi_j \Lambda^{-1})^{-1}$. where $\hat{\Phi}$ contains only the features that are currently included in the model, and $\hat{m}_w$ and $\hat{\mathbf{V}}_w$ are obtained based on these features. This observation allows us to efficiently compute the EP approximation and update $\boldsymbol{\alpha}$, since in general there are only a small set of the features in the model during the updates.

4.3.4 Algorithm summary

To summarize the **predictive-ARD-EP** algorithm:

1. First, initialize the model so that it only contains a small fraction of features.

2. Then, sequentially update $\boldsymbol{\alpha}$ as in section 4.3.3 and calculate the required statistics by EP as in section 4.3.1 until the algorithm converges.

3. Finally, choose the classifier from the sequential updates with minimum leave-one-out error estimate (4.12). The leave-one-out error is discrete, so in case of a tie, choose the first classifier in the tie, i.e., the one with the smaller evidence.

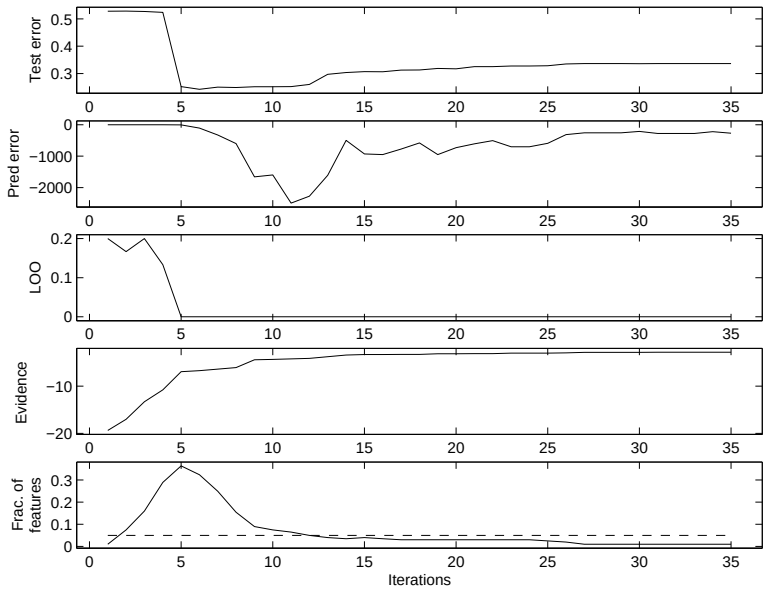
A variant of this algorithm uses the error probability (4.13) and is called **predictive**$_{Prob}$**-ARD-EP**. Choosing the classifier with the maximum evidence (approximated by EP) is called **evidence-ARD-EP**.

4.4 Experiments and discussion

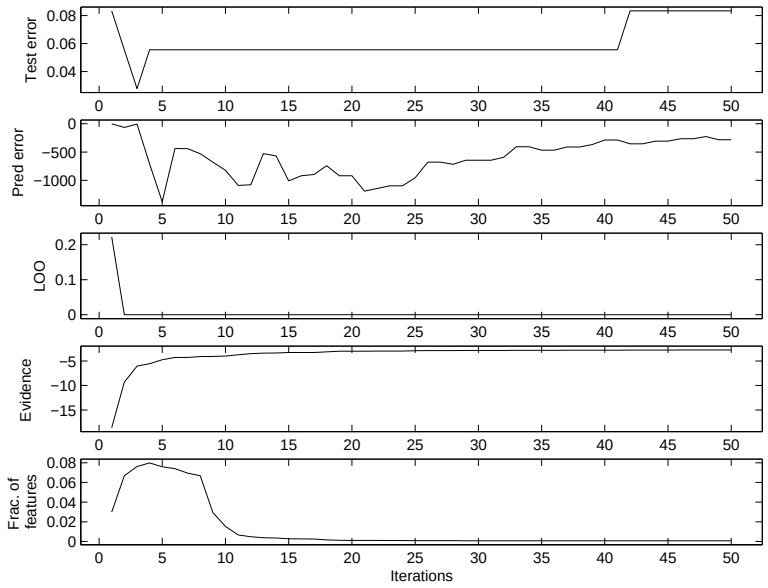
This section compares evidence-ARD-EP and predictive-ARD-EP on synthetic and real-world data sets. The first experiment has 30 random training points and 5000 random test points with dimension 200. True classifier weights consist of 10 Gaussian weights, sampled from $\mathcal{N}(-0.5, 1)$ and 190 zero weights. The true classifier is then used as the ground truth to label the data. Thus the data is guaranteed to be separable. The basis functions are simply $\boldsymbol{\phi}(\mathbf{x}) = \mathbf{x}$. The results over 50 repetitions of this procedure are visualized in Figure 4-4-(a). Both predictive-ARD-EP and predictive$_{Prob}$-ARD-EP outperform evidence-ARD-EP by picking the model with the smallest estimate of the predictive error, rather than choosing the most probable model.

Figure 4-3-(a) shows a typical run. As shown in the Figure, the estimates of the predictive performance based on leave-one-out error count (4.12) and (log) error probability (4.13) are better correlated with the true test error than evidence and the fraction of features. The evidence is computed as in equation (4.11) and the fraction of features is defined as $\frac{||\mathbf{w}||_0}{d}$ where d is the dimension of the classifier $\mathbf{w}$. Also, since the data is designed to be linearly separable, we always get zero training error along the iterations. While the (log) evidence keeps increasing, the test error rate first decreases and then increases. This demonstrates the overfitting problem associated with maximizing evidence. As to the fraction of features, it first increases by adding new useful features into the model and then decreases by deleting old features. Notice that the fraction of features converges to a lower value than the true one, $\frac{10}{200} = 0.05$, which is plotted as the dashed line in Figure4-3-(a). Moreover, choosing the sparsest model, i.e., the one with the smallest fraction of features, leads to overfitting here even though there is zero training error.

Next, the algorithms are applied to high-dimensional gene expression datasets: leukaemia and colon cancer. For the leukaemia dataset, the task is to distinguish acute myeloid leukaemia (AML) from acute lymphoblastic leukaemia (ALL). The dataset has 47 and 25 samples of type ALL and AML respectively with 7129 features per sample. The dataset was randomly split 100 times into 36 training and 36 test samples, with evidence-ARD-EP and predictive-ARD-EP run on each. Figure 4-3-(b) shows a typical run that again illustrates the overfitting phenomenon as shown in Figure 4-3-(a). On most of runs including the one shown in Figure 4-3-(b), there is zero training error from the first to the last iterations.



(a) Synthetic data classification

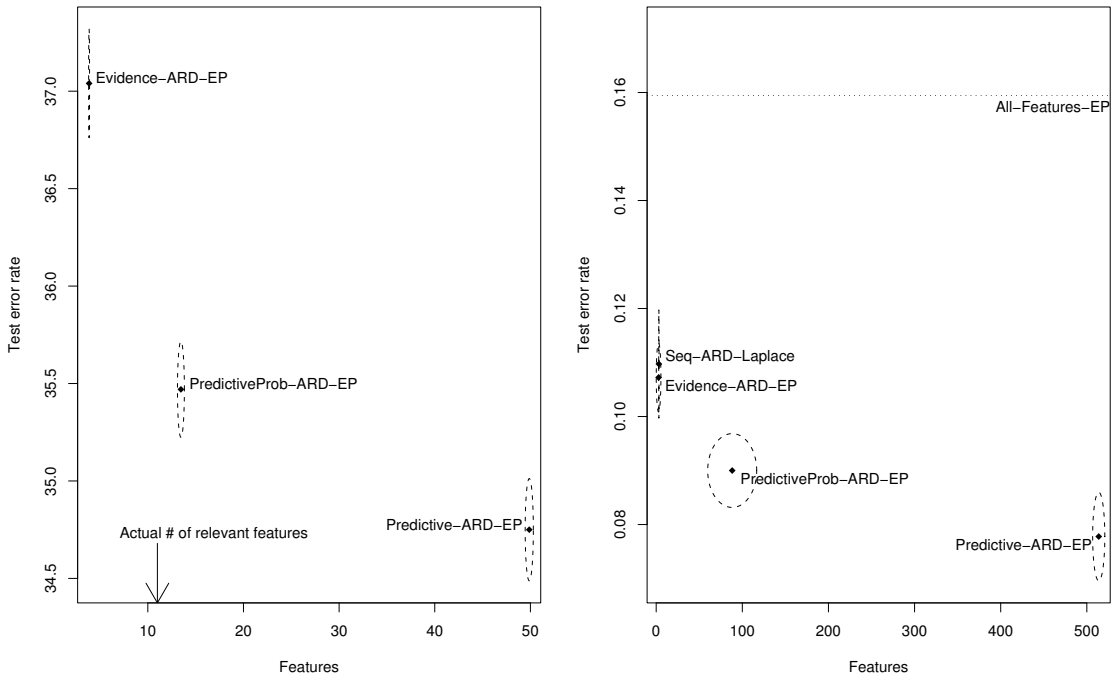


(b) Leukaemia data classification

Figure 4-3: Comparison of different model selection criteria during ARD training. The second and third rows are computed via (4.13) and (4.12), respectively. The estimated predictive performance is better correlated with the test errors than evidence and sparsity.

The test performance is visualized in Figure 4-4-(b). The error counts of evidence-ARD-EP and predictive-ARD-EP are 3.86 ± 0.14 and 2.80 ± 0.18, respectively. The numbers of the chosen features of these two methods are 2.78 ± 1.65 and 513.82 ± 4.30, respectively.

For the colon cancer dataset, the task is to discriminate tumour from normal tissues using microarray data. The whole dataset has 22 normal and 40 cancer samples with



(a) Synthetic data classification (b) Leukaemia data classification

Figure 4-4: Test errors and sizes of selected features. (a) synthetic dataset with 30 training and 5000 test data points. Each data point has 201 features, among which only 11 are useful. 50 random repetitions of the data were used. (b) leukaemia microarray dataset with 36 training and 36 test data points. Each data point has 7129 features. The results are averaged over 100 random partitions. Ellipses indicate standard errors over repetitions.

2000 features per sample. We randomly split the dataset into 50 training and 12 test samples 100 times and run evidence-ARD-EP and predictive-ARD-EP on each partition. The test performance is visualized in Figure 4-5. For comparison, we show the results from Li et al. (2002). The methods tested by Li et al. (2002) include ARD-Laplace with fast sequential updates on a logistic model (Seq-ARD-Laplace), Support Vector Machine (SVM) with recursive feature elimination (SVM-RFE), and SVM with Fisher score feature ranking (SVM-Fisher Score). The error counts of evidence-ARD-EP and predictive-ARD-EP are 2.54 ± 0.13 and 1.63 ± 0.11, respectively. The sizes of chosen feature sets for these two methods are 7.92 ± 0.14 and 156.76 ± 11.68, respectively.

As shown in figures 4-4-(b) and 4-5, pruning irrelevant features by maximizing evidence helps to reduce test errors. But aggressive pruning will overfit the model and therefore increase the test errors. For both colon cancer and leukaemia datasets, predictive-ARD-EP with a moderate number of features outperforms all the other methods including EP without feature pruning as well as the evidence-ARD-EP with only a few features left in

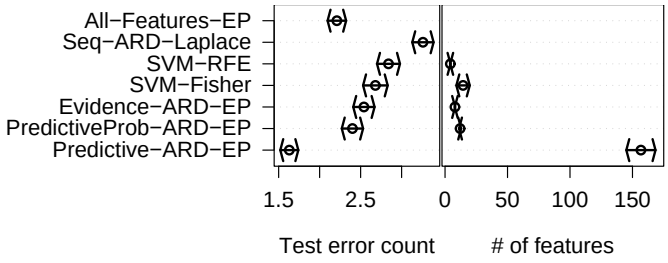


Figure 4-5: Test errors and sizes of selected feature sets on colon cancer microarray dataset with 50 training and 12 test data points. Each data point has 2000 features. 100 random partitionings of the data were used.

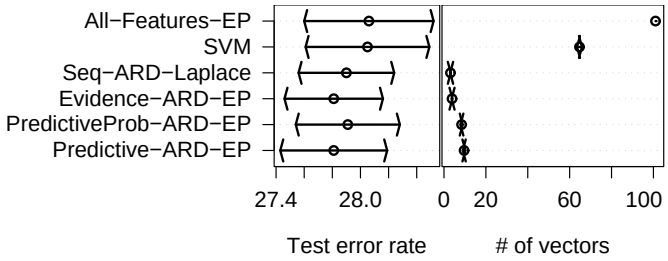


Figure 4-6: Test error rates and numbers of relevance or support vectors on breast cancer dataset. 50 partitionings of the data were used. All these methods use the same Gaussian kernel with kernel width $\sigma = 5$. The trade-off parameter C in SVM is chosen via 10-fold cross-validation for each partition.

the model.

Finally, RBF-type feature expansion can be combined with predictive-ARD-EP to obtain sparse nonlinear Bayesian classifiers. Specifically, we use the following Gaussian basis function $\phi(\mathbf{x}_i)$

$$\phi(\mathbf{x}_i) = [\mathbf{1}, k(\mathbf{x}_i, \mathbf{x}_1), \ldots, k(\mathbf{x}_i, \mathbf{x}_N)]^{\mathrm{T}}$$

where $k(\mathbf{x}_i, x_j) = \exp(-\frac{||\mathbf{x}_i - \mathbf{x}_j||^2}{2\sigma^2})$. Unlike the ARD feature selection in the previous examples, here ARD is used to choose relevance vectors $\phi(\mathbf{x}_i)$, i.e., to select data points instead of features. The algorithms are tested on two UCI datasets: breast cancer and diabetes.

For the breast cancer dataset provided by Zwitter and Soklic (1998), the task is to distinguish no-recurrence-events from recurrence-events. We split the dataset into 100 training and 177 test samples 50 times and run evidence-ARD-EP and predictive-ARD-EP on each partition. The test performance is visualized in Figure 4-6 and summarized in Table 4.1.

In this experiment, evidence-ARD-EP and predictive-ARD-EP marginally outperform the other alternatives, but give much simpler models. They only have about 4 or 10 relevance vectors while SVM uses about 65 support vectors.

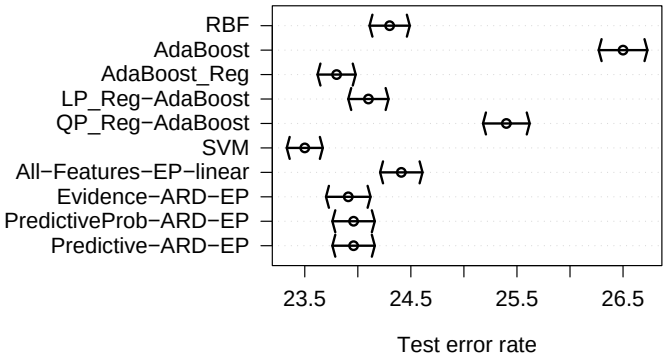


Figure 4-7: Test errors on diabetes dataset with 468 training and 300 test data points. The results are averaged over 100 partitions.

Finally evidence-ARD-EP and predictive-ARD-EP are tested on the UCI diabetes dataset. Each is run on 100 random partitions of 468 training and 300 test samples. The partitions are the same as in Rätsch et al. (2001), so that we can directly compare our results with theirs. The test performance is summarized in Figure 4.4. The error rates of the evidence-ARD-EP and predictive-ARD-EP are $23.91 \pm 0.21\%$ and $23.96 \pm 0.20\%$, respectively.

On the diabetes dataset, predictive-ARD-EP performs comparably or outperforms most of the other state-of-the-art methods; only SVM performs better. Note that the SVM's kernel has been optimized using the test data points, which is not possible in practice (Rätsch et al., 2001).

Table 4.1: Test error rates and numbers of relevance or support vectors on breast cancer dataset.

Algorithm	Test error rate (%)	Size of feature set
SVM	28.05 ± 0.44	64.70 ± 1.17
EP	28.06 ± 0.46	101 ± 0
Seq-ARD-Laplace	27.90 ± 0.34	3.10 ± 0.13
Evd-ARD-EP	$\mathbf{27.81 \pm 0.35}$	3.84 ± 0.19
Pred_{Prob}-ARD-EP	27.91 ± 0.37	8.40 ± 0.54
Pred-ARD-EP	$\mathbf{27.81 \pm 0.38}$	9.60 ± 0.62

4.5 Conclusions

This chapter has presented predictive-ARD-EP, an efficient algorithm for feature selection and sparse learning. Predictive-ARD-EP chooses the model with the best estimate of the predictive performance instead of choosing the one with the largest marginal likelihood. On high-dimensional microarray datasets, predictive-ARD-EP outperforms other state-of-the-art algorithms in test accuracy. On UCI benchmark datasets, it results in sparser classifiers than SVMs with comparable test accuracy. The resulting sparse models can be used in applications where classification time is critical. To achieve a desired balance between test accuracy and testing time, one can choose a classifier that minimizes a loss function trading off the leave-one-out error estimate and the number of features.

The success of this algorithm argues against a few popular principles in learning theory. First, it argues against the evidence framework in which the evidence is maximized by tuning hyperparameters. Maximizing evidence is useful for choosing among a small set of models, but can overfit if used with a large continuum of models, as in ARD. Second, our findings show that larger fraction of nonzero features (or lower sparsity) can lead to better generalization performance, even when the training error is zero. This is against the sparsity principles as well as Occam's razor. If what we care about is generalization performance, then it is better to minimize some measure of predictive performance as predictive ARD does.

Chapter 5

Bayesian conditional random fields

5.1 Introduction

Traditional classification models, including the probit model used in Chapter 4, often assume that data items are independent. However, real world data is often interdependent and has complex structure. Suppose we want to classify web pages into different categories, e.g., homepages of students versus faculty. The category of a web page is often related to the categories of pages linked to it. Rather than classifying pages independently, we should model them jointly to incorporate such contextual cues.

Joint modeling of structured data can be performed by generative graphical models, such as Bayesian networks or Markov random fields. For example, hidden Markov models have been used in natural language applications to assign labels to words in a sequence, where labels depend both on words and other labels along a chain. However, generative models have fundamental limitations. Firstly, generative models require specification of the data generation process, i.e., how data can be sampled from the model. In many applications, this process is unknown or impractical to write down, and not of interest for the classification task. Secondly, generative models typically assume conditional independence of observations given the labels. This independence assumption limits their modeling power and restricts what features can be extracted for classifying the observations. In particular, this assumption rules out features capturing long-range correlations, multiple scales, or other context.

Conditional random fields (CRF) are a conditional approach for classifying structured

[1]This chapter includes joint work with Martin Szummer and Thomas P. Minka.

data, proposed by Lafferty et al. (2001). CRFs model only the label distribution conditioned on the observations. Unlike generative models, they do not need to explain the observations or features, and thereby conserve model capacity and reduce effort. This also allows CRFs to use flexible features such as complex functions of multiple observations. The modeling power of CRFs has shown great benefit in several applications, such as natural language parsing (Sha & Pereira, 2003), information extraction (McCallum, 2003), and image modeling (Kumar & Hebert, 2004).

To summarize, CRFs provide a compelling model for structured data. Consequently, there has been an intense search for effective training and inference algorithms. The first approaches maximized conditional likelihood (ML), either by generalized iterative scaling or by quasi-Newton methods (Lafferty et al., 2001; Sha & Pereira, 2003). However, the ML criterion is prone to overfitting the data, especially since CRFs are often trained with very large numbers of correlated features. The maximum a posteriori (MAP) criterion can reduce overfitting, but provides no guidance on the choice of parameter prior. Furthermore, large margin criteria have been applied to regularize the model parameters and also to kernelize CRFs (Taskar et al., 2004; Lafferty et al., 2004). Nevertheless, training and inference for CRFs remain challenges, and the problems of overfitting, feature and model selection have largely remained open.

In this chapter, we propose Bayesian Conditional Random Fields (BCRF), a novel Bayesian approach to training and inference for conditional random fields. Applying the Bayesian framework brings principled solutions and tools for addressing overfitting, model selection and many other aspects of the problem. Unlike ML, MAP, or large-margin approaches, we train BCRFs by estimating the posterior distribution of the model parameters. Subsequently, we can can average over the posterior distribution for BCRF inference.

The complexity of the partition function in CRFs (the denominator of the likelihood function) necessitates approximations. While traditional EP and variational methods can not be directly applied, because of the presence of partition functions, the power EP (PEP) method (Minka, 2004) can be used for BCRF training. However, PEP can lead to convergence problems. To solve these problems, this chapter proposes the transformed PEP method to incorporate partition functions in BCRF training. Furthermore, we flatten the approximation structures to avoid two-level approximations. This significantly enhances the algorithmic stability and improves the estimation accuracy. For computational efficiency,

we use low-rank matrix operations.

This chapter first formally defines CRFs, presents the power EP and the new transformed PEP methods, and flattens the approximation structure for training. Then it proposes an approximation method for model averaging, and finally shows experimental results.

5.2 From conditional random fields to BCRFs

A conditional random field (CRF) models label variables according to an undirected graphical model conditioned on observed data (Lafferty et al., 2001). Let $\mathbf{x}$ be an "input" vector describing the observed data instance, and $\mathbf{t}$ be an "output" random vector over labels of the data components. We assume that all labels for the components belong to a finite label alphabet $\mathcal{T} = \{1, \ldots, T\}$. For example, the input $\mathbf{x}$ could be image features based on small patches, and $\mathbf{t}$ be labels denoting 'person, 'car' or 'other' patches. Formally, we have the following definition of CRFs (Lafferty et al., 2001):

Definition 5.2.1 *Let $G = (\mathcal{V}, \mathcal{E})$ be a graph such that $\mathbf{t}$ is indexed by the vertices of G. Then $(\mathbf{x}, \mathbf{t})$ is a conditional random field (CRF) if, when conditioned on $\mathbf{x}$, the random variables $\{t_i\}$ obey the Markov property with respect to the graph: $p(t_i|\mathbf{x}, \mathbf{t}_{\mathcal{V}-i}) = p(t_i|\mathbf{x}, \mathbf{t}_{\mathcal{N}_i})$ where $\mathcal{V}-i$ is the set of all nodes in G except the node i, $\mathcal{N}_i$ is the set of neighbors of the node i in G, and $\mathbf{t}_\Omega$ represents the random variables of the vertices in the set Ω.*

Unlike traditional generative random fields, CRFs only model the conditional distribution $p(\mathbf{t}|\mathbf{x})$ and do not explicitly model the marginal $p(\mathbf{x})$. Note that the labels $\{t_i\}$ are globally conditioned on the whole observation $\mathbf{x}$ in CRFs. Thus, we do not assume that the observed data $\mathbf{x}$ are conditionally independent as in a generative random field.

BCRFs are a Bayesian approach to training and inference with conditional random fields. In some sense, BCRFs can be viewed as an extension of conditional Bayesian linear classifiers, e.g., Bayes point machines (BPM) (Herbrich et al., 1999; Minka, 2001), which are used to classify independent data points.

According to the Hammersley-Clifford theorem, a CRF defines the conditional distribution of the labels $\mathbf{t}$ given the observations $\mathbf{x}$ to be proportional to a product of potential functions on cliques of the graph G. For simplicity, we consider only pairwise clique poten-

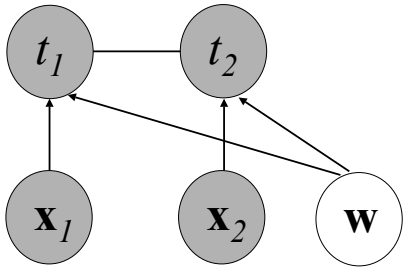


Figure 5-1: Graphical model for Bayesian conditional random field. The shaded nodes represent observed variables, i.e., two data points linked with each other, and the unshaded node represents latent variables, i.e., the model parameters $\mathbf{w}$. As a Bayesian approach, BCRFs estimate the posterior distribution of the parameters $\mathbf{w}$ in the graph. Compared to Figure 4-1, we can clearly see the model difference between BCRFs and traditional Bayesian classification.

tials such that

$$p(\mathbf{t}|\mathbf{x}, \mathbf{w}) = \frac{1}{Z(\mathbf{w})} \prod_{\{i,j\}\in\mathcal{E}} g_{i,j}(t_i, t_j, \mathbf{x}; \mathbf{w}) \tag{5.1}$$

where

$$Z(\mathbf{w}) = \sum_{\mathbf{t}} \prod_{\{i,j\}\in\mathcal{E}} g_{i,j}(t_i, t_j, \mathbf{x}; \mathbf{w}) \tag{5.2}$$

is a normalizing factor known as the partition function, $g_{i,j}(t_i, t_j, \mathbf{x}; \mathbf{w})$ are pairwise potentials, and $\mathbf{w}$ are the model parameters. Note that the partition function is a complicated function of the model parameters $\mathbf{w}$. This makes Bayesian training much harder for CRFs than for Bayesian linear classifiers, since the normalizer of a Bayesian linear classifier is a constant. A simple CRF is illustrated in Figure 5-1. As a Bayesian approach, BCRFs estimate the posterior distribution of the parameters $\mathbf{w}$, the unshaded node in the graphical model.

In standard conditional random fields, the pairwise potentials are defined as

$$g_{i,j}(t_i, t_j, \mathbf{x}; \mathbf{w}) = \exp(\mathbf{w}_{t_i,t_j}^{\mathrm{T}} \boldsymbol{\phi}_{i,j}(t_i, t_j, \mathbf{x})) \tag{5.3}$$

where $\boldsymbol{\phi}_{i,j}(t_i, t_j, \mathbf{x})$ are features extracted for the edge between vertices i and j of the conditional random field, and $\mathbf{w}_{t_i,t_j}$ are elements corresponding to labels $\{t_i, t_j\}$ in $\mathbf{w}$, where $\mathbf{w} = [\mathbf{w}_{1,1}^{\mathrm{T}}, \mathbf{w}_{1,2}^{\mathrm{T}}, \ldots, \mathbf{w}_{T,T}^{\mathrm{T}}]^{\mathrm{T}}$. There are no restrictions on the relation between features.

Though we could use the above log-linear potential functions, a Bayesian treatment would require additional approximation techniques, e.g., quadrature or Monte Carlo sampling. More conveniently, we replace the exponential by the probit function $\Psi(\cdot)$ (the cumulative distribution function of a Gaussian with mean 0 and variance 1). This permits analytic Bayesian training and inference in BCRFs. In addition, we wish to incorporate robustness against labeling errors, and assume a small probability ϵ of a label being incorrect, which will serve to bound the potential value away from zero. Specifically, our robust potentials are

$$g_{i,j}(t_i, t_j, \mathbf{x}; \mathbf{w}) = (1-\epsilon)\Psi(\mathbf{w}_{t_i,t_j}^{\mathrm{T}} \boldsymbol{\phi}_{i,j}(t_i, t_j, \mathbf{x})) + \epsilon(1 - \Psi(\mathbf{w}_{t_i,t_j}^{\mathrm{T}} \boldsymbol{\phi}_{i,j}(t_i, t_j, \mathbf{x}))). \tag{5.4}$$

5.3 Training BCRFs

Given the data likelihood and a Gaussian prior

$$p_0(\mathbf{w}) \sim \mathcal{N}(\mathbf{w}|\mathbf{0}, \mathrm{diag}(\boldsymbol{\alpha})),$$

the posterior of the parameters is

$$p(\mathbf{w}|\mathbf{t}, \mathbf{x}) \propto \frac{1}{Z(\mathbf{w})} p_0(\mathbf{w}) \prod_{\{k\} \in \mathcal{E}} g_k(t_i, t_j, \mathbf{x}; \mathbf{w})$$

where $k = i, j$ indexes edges in a CRF.

Expectation propagation exploits the fact that the posterior is a product of simple terms. If we approximate each of these terms well, we can get a good approximation of the posterior. Mathematically, EP approximates $p(\mathbf{w}|\mathbf{t}, \mathbf{x})$ as

$$q(\mathbf{w}) = p_0(\mathbf{w}) \frac{1}{\tilde{Z}(\mathbf{w})} \prod_{\{k\} \in \mathcal{E}} \tilde{g}_k(\mathbf{w}) \tag{5.5}$$

$$= \frac{1}{\tilde{Z}(\mathbf{w})} \tilde{R}(\mathbf{w}) \tag{5.6}$$

where $\tilde{R}(\mathbf{w}) = p_0(\mathbf{w}) \prod_{\{k\} \in \mathcal{E}} \tilde{g}_k(\mathbf{w})$ is the numerator in the approximate posterior distribution $q(\mathbf{w})$. The approximation terms $\tilde{g}_k(\mathbf{w})$ and $\frac{1}{\tilde{Z}(\mathbf{w})}$ have the form of a Gaussian, so that the approximate posterior $q(\mathbf{w})$ is a Gaussian, i.e., $q(\mathbf{w}) \sim \mathcal{N}(\mathbf{m}_w, \boldsymbol{\Sigma}_w)$. We can obtain

the approximation terms $\tilde{g}_k(\mathbf{w})$ in the same way as in Bayesian linear classifiers (Minka, 2001) Specifically, for the g_k term, the algorithm first computes $q^{\backslash k}(\mathbf{w})$, which represents the "rest of the distribution." Then it minimizes KL-divergence over $\tilde{g}_k$, holding $q^{\backslash k}$ fixed. This process can be written succinctly as follows:

$$q^{\backslash k}(\mathbf{w}) \propto q(\mathbf{w})/\tilde{g}_k(\mathbf{w}) \tag{5.7}$$

$$\tilde{g}_k(\mathbf{w})^{new} = \underset{\acute{g}_k(\mathbf{w})}{\operatorname{argmin}} \operatorname{KL}(g_k(\mathbf{w})q^{\backslash k}(\mathbf{w}) \;||\; \acute{g}_k(\mathbf{w})q^{\backslash k}(\mathbf{w})) \tag{5.8}$$

$$= \operatorname{proj}\Big[g_k(\mathbf{w})q^{\backslash k}(\mathbf{w})\Big] / q^{\backslash k}(\mathbf{w}) \tag{5.9}$$

$$q(\mathbf{w})^{new} = q^{\backslash k}(\mathbf{w})\tilde{g}_k(\mathbf{w})^{new} \tag{5.10}$$

where proj is a "moment matching" operator: it finds the Gaussian having the same moments as its argument, thus minimizing KL. Algorithmically, (5.7) means "divide the Gaussians to get a new Gaussian, and call it $q^{\backslash k}(\mathbf{w})$." Similarly, (5.9) means construct a Gaussian whose moments match $g_k(\mathbf{w})q^{\backslash k}(\mathbf{w})$ and divide it by $q^{\backslash k}(\mathbf{w})$, to get a new Gaussian approximation term $\tilde{g}_k(\mathbf{w})^{new}$." These are the updates to incorporate the exact terms in the numerator. The main difficulty for BCRF training is how to approximate the denominator $1/Z(\mathbf{w})$ and incorporate its approximation into $q(\mathbf{w})$.

5.3.1 Power EP and Transformed PEP

This section presents the power EP (PEP) and the transformed PEP methods to approximate an integral involving the denominator $1/Z(\mathbf{w})$. Traditional EP has difficulty to incorporate the denominator, since it is hard to directly compute required moments for the KL minimization in the projection step. The PEP and the transformed PEP methods solve this problem by changing the projection step.

Power EP

The first method is the power EP method, which is an extension of EP to make the approximation more tractable. It was first used by Minka and Lafferty (2002), in the case of having terms with positive powers. However, PEP also works with negative powers, and this is one of the key insights that makes approximate BCRF training tractable.

Specifically, to incorporate a term $s_k(\mathbf{w})$ approximated by $\tilde{s}_k(\mathbf{w})$, PEP extends EP by

using the following updates with a desired power $n_k \neq 0$:

$$q^{\backslash k}(\mathbf{w}) \propto q(\mathbf{w})/\tilde{s}_k(\mathbf{w})^{1/n_k} \tag{5.11}$$

$$\tilde{s}_k(\mathbf{w})^{new} = \left(\text{proj}\Big[s_k(\mathbf{w})^{1/n_k} q^{\backslash k}(\mathbf{w})\Big] / q^{\backslash k}(\mathbf{w})\right)^{n_k} \tag{5.12}$$

$$q(\mathbf{w})^{new} = q(\mathbf{w})\frac{\tilde{s}_k(\mathbf{w})^{new}}{\tilde{s}_k(\mathbf{w})} = q^{\backslash k}(\mathbf{w})\frac{\tilde{s}_k(\mathbf{w})^{new}}{\tilde{s}_k(\mathbf{w})^{1-1/n_k}} \tag{5.13}$$

As shown by Minka (2004), these updates seek to minimize a different measure of divergence, the α-divergence, where $\alpha = 2/n_k - 1$. Note that α can be any real number. By picking n_k appropriately, the updates can be greatly simplified, while still minimizing a sensible error measure. This result is originally from Wiegerinck and Heskes (2002). They discussed an algorithm called "fractional belief propagation" which is a special case of PEP, and all of their results also apply to PEP.

For BCRF training, we will use these PEP updates for the denominator term $s_k(\mathbf{w}) = 1/Z(\mathbf{w})$ by picking $n_k = -1$. Specifically, we have the following updates.

1. Deletion: we compute the "leave-one-out" approximate posterior $q^{\backslash z}(\mathbf{w})$, by removing the old approximation of the partition function $\tilde{Z}(\mathbf{w})$ from the current approximation $q(\mathbf{w})$ as if it were a numerator.

$$q^{\backslash z}(\mathbf{w}) \propto \frac{q(\mathbf{w})}{\tilde{Z}(\mathbf{w})} = \frac{1}{(\tilde{Z}(\mathbf{w}))^2}\tilde{R}(\mathbf{w}) \tag{5.14}$$

2. Projection: given the "leave-one-out" approximate posterior $q^{\backslash z}(\mathbf{w})$, we compute the new "exact" posterior:

$$\tilde{q}(\mathbf{w}) = \text{proj}\Big[Z(\mathbf{w})q^{\backslash z}(\mathbf{w})\Big] = \underset{\tilde{q}(\mathbf{w})}{\text{argmin}}\, \text{KL}(q^{\backslash z}(\mathbf{w})Z(\mathbf{w})||\tilde{q}(\mathbf{w}))$$

by matching the moments of $\tilde{q}(\mathbf{w})$ and $q^{\backslash z}(\mathbf{w})Z(\mathbf{w})$). Thanks to picking $n_k = -1$, we need only the moments of $Z(\mathbf{w})q^{\backslash k}(\mathbf{w})$, instead of those of $\frac{q^{\backslash k}(\mathbf{w})}{Z(\mathbf{w})}$. Since we cannot compute the moments of $Z(\mathbf{w})q^{\backslash k}(\mathbf{w})$ analytically, we embed an EP algorithm to approximate these moments (see details in Section 5.3.2). Note that we cannot apply EP directly to approximate the moments of $\frac{q^{\backslash k}(\mathbf{w})}{Z(\mathbf{w})}$ without resorting to other approximation methods, such as quadrature or Monte Carlo.

Given $\tilde{q}(\mathbf{w})$, we update $\tilde{Z}(\mathbf{w})^{new}$ as follows:

$$\tilde{Z}(\mathbf{w})^{new} \propto \tilde{q}(\mathbf{w})/q^{\backslash z}(\mathbf{w}),$$

3. Inclusion: we replace the old approximation term $\tilde{Z}(\mathbf{w})$ with a new one to obtain $q(\mathbf{w})^{new}$.

$$q(\mathbf{w})^{new} = q^{\backslash z}(\mathbf{w})\frac{(\tilde{Z}(\mathbf{w}))^2}{\tilde{Z}(\mathbf{w})^{new}} \tag{5.15}$$

Transformed PEP

PEP has feedback loops in its updates: the "leave-one-out" approximate posterior $q^{\backslash k}(\mathbf{w})$ involves the old approximation term $\tilde{s}_k(\mathbf{w})$. This is because we remove only a fraction of the old approximation term from $q(\mathbf{w})$ in the deletion step:

$$q^{\backslash k}(\mathbf{w}) = q(\mathbf{w})/\tilde{s}_k(\mathbf{w})^{1/n_k} = \tilde{s}_k(\mathbf{w})^{1-1/n_k}\prod_{j\neq k}\tilde{s}_j(\mathbf{w})$$

Therefore, the new approximation term based on $q^{\backslash k}(\mathbf{w})$ is dependent on the old approximation term. This dependency can cause oscillations in training. To address this problem and obtain faster convergence speeds, we weaken the feedback loops in PEP by introducing a damping factor m_k in the deletion step. The damping factor transforms the power of the approximation term in the deletion step of PEP. Thus, we name this method transformed PEP (TPEP). Specifically, to incorporate the term $s_k(\mathbf{w})$ approximated by $\tilde{s}_k(\mathbf{w})$, transformed PEP generalizes PEP by using the following updates:

$$q^{\backslash k}(\mathbf{w}) \propto q(\mathbf{w})/\tilde{s}_k(\mathbf{w})^{1-m_k+\frac{m_k}{n_k}} \tag{5.16}$$

$$\tilde{s}_k(\mathbf{w})^{new} = \left(\text{proj}\Big[s_k(\mathbf{w})^{1/n_k}q^{\backslash k}(\mathbf{w})\Big]/q^{\backslash k}(\mathbf{w})\right)^{n_k} \tag{5.17}$$

$$q(\mathbf{w})^{new} = q(\mathbf{w})\frac{\tilde{s}_k(\mathbf{w})^{new}}{\tilde{s}_k(\mathbf{w})} = q^{\backslash k}(\mathbf{w})\frac{\tilde{s}_k(\mathbf{w})^{new}}{\tilde{s}_k(\mathbf{w})^{m_k-m_k/n_k}} \tag{5.18}$$

where $m_k \in [0,1]$. If $m_k = 1$, then transformed PEP reduces to PEP. If $m_k = 0$, then the "leave-one-out" approximate posterior does not involve any information in the old approximation term $\tilde{s}_k(\mathbf{w})$. We name this special case the transformed EP (TEP) method. TEP essentially cuts the feedback loops in PEP. By sharing the same inclusion and deletion

steps, TEP is more similar to EP than PEP is to EP. TEP differs from EP only in the projection step by transforming the exact and the approximation terms in EP. Hence the name "transformed EP". In practice, TEP tends to have faster convergence rates than PEP, while sacrificing accuracy to some extent. We can also choose m_k between 0 and 1 to damp the feedback loops and obtain a tradeoff between PEP and TEP.

For BCRF training, we will use the above updates for the denominator term $s_k(\mathbf{w}) = 1/Z(\mathbf{w})$. We pick $n_k = -1$ and $m_k = 0$ such that transformed PEP is reduced to TEP:

1. Deletion: $q^{\backslash z}(\mathbf{w}) \propto q(\mathbf{w})\tilde{Z}(\mathbf{w}) = \tilde{R}(\mathbf{w})$. Here $q^{\backslash z}(\mathbf{w})$ does not involve the old term approximation $\tilde{Z}(\mathbf{w})$, while $q^{\backslash z}(\mathbf{w})$ in PEP does.

2. Projection: $\tilde{q}(\mathbf{w}) = \text{proj}\big[Z(\mathbf{w})q^{\backslash z}(\mathbf{w})\big]$.

   Again, as in the PEP method, we need to use another EP to approximate the required moments. Given the new moments of $\tilde{q}(\mathbf{w})$, we update the new approximation term $\tilde{Z}(\mathbf{w})^{new} \propto \tilde{q}(\mathbf{w})/q^{\backslash z}(\mathbf{w})$.

3. Inclusion: $q(\mathbf{w})^{new} = \frac{q^{\backslash z}(\mathbf{w})}{\tilde{Z}(\mathbf{w})^{new}}$.

5.3.2 Approximating the partition function

In the moment matching step, we need to approximate the moments of $Z(\mathbf{w})q^{\backslash z}(\mathbf{w})$ where $Z(\mathbf{w})$ is the exact partition function, since $Z(\mathbf{w})$ is a very complicated function of $\mathbf{w}$. Murray and Ghahramani (2004) have proposed approximate MCMC methods to approximate the partition function of an undirected graph. This section presents an alternative method.

For clarity, let us rewrite the partition function as follows:

$$Z(\mathbf{w}) = \sum_{\mathbf{t}} Z(\mathbf{w}, \mathbf{t}) \tag{5.19}$$

$$Z(\mathbf{w}, \mathbf{t}) = \prod_{k \in \mathcal{E}} g_k(t_i, t_j, \mathbf{x}; \mathbf{w}) \tag{5.20}$$

where $k = \{i, j\}$ indexes edges. To compute the moments of $Z(\mathbf{w})q^{\backslash z}(\mathbf{w})$, we can embed EP. Specifically, the EP approximation factorizes over w and $\mathbf{t}$:

$$\tilde{q}(\mathbf{w}) = \text{proj}\Big[Z(\mathbf{w})q^{\backslash z}(\mathbf{w})\Big] \tag{5.21}$$

$$\tilde{q}(\mathbf{w})q(\mathbf{t}) = \tilde{Z}(\mathbf{w})\tilde{Z}(\mathbf{t})\tilde{q}^{\backslash z}(\mathbf{w}) \tag{5.22}$$

$$\tilde{Z}(\mathbf{w})\tilde{Z}(\mathbf{t}) = \prod_{k\in\mathcal{E}} \tilde{f}_k(\mathbf{w})\tilde{f}_k(t_i)\tilde{f}_k(t_j) \tag{5.23}$$

where $q(\mathbf{t})$ is the approximate distribution for labels, and $\tilde{f}_k(\mathbf{w})\tilde{f}_k(t_i)\tilde{f}_k(t_j)$ approximates $g_k(t_i, t_j, \mathbf{x}; \mathbf{w})$ in the denominator.

Because the approximation is factorized, the computation will have the flavor of loopy belief propagation. The initial $\tilde{f}_k$'s will all be 1, making the initial $\tilde{q}(\mathbf{w}) = q^{\setminus z}(\mathbf{w})$ and $q(\mathbf{t}) = 1$. The EP updates for the $\tilde{f}_k$'s are:

$$\tilde{q}^{\setminus k}(\mathbf{w}) \propto \tilde{q}(\mathbf{w})/\tilde{f}_k(\mathbf{w}) \tag{5.24}$$

$$q^{\setminus k}(t_i) \propto q(t_i)/\tilde{f}_k(t_i) \qquad \text{(similarly for } j\text{)} \tag{5.25}$$

$$f_k(\mathbf{w}) = \sum_{t_i,t_j} g_k(t_i, t_j, \mathbf{x}; \mathbf{w}) q^{\setminus k}(t_i) q^{\setminus k}(t_j) \tag{5.26}$$

$$\tilde{f}_k(\mathbf{w})^{new} = \text{proj}\Big[f_k(\mathbf{w})\tilde{q}^{\setminus k}(\mathbf{w})\Big] / \tilde{q}^{\setminus k}(\mathbf{w}) \tag{5.27}$$

$$\tilde{f}_k(t_i)^{new} = \sum_{t_j} \int_{\mathbf{w}} g_k(t_i, t_j, \mathbf{x}; \mathbf{w}) \tilde{q}^{\setminus k}(\mathbf{w}) q^{\setminus k}(t_j) d\mathbf{w} \tag{5.28}$$

$$\tilde{q}(\mathbf{w})^{new} = \tilde{q}^{\setminus k}(\mathbf{w})\tilde{f}_k(\mathbf{w})^{new} \tag{5.29}$$

$$q(t_i)^{new} = q^{\setminus k}(t_i)\tilde{f}_k(t_i)^{new} \tag{5.30}$$

These updates are iterated for all k, until a fixed point is reached. A straightforward implementation of the above updates costs $O(d^3)$ time, where d is the dimension of the parameter vector $\mathbf{w}$, since the covariance matrix of $\mathbf{w}$ must be inverted. However, as shown in the next section, it is possible to compute them with low-rank matrix updates, in $O(d^2)$ time.

5.3.3 Efficient low-rank matrix computation

This section presents low-rank matrix computation for the new approximate posterior distributions, the new approximation terms, and the "leave-one-out" posterior distributions. These calculations are embedded in the projection step of either PEP or TPEP updates.

Computing new approximate posterior

First, let us define $\boldsymbol{\phi}_k(m, n, \mathbf{x})$ as shorthand of $\boldsymbol{\phi}_k(t_i = m, t_j = n, \mathbf{x})$, where $\boldsymbol{\phi}_k(t_i, t_j, \mathbf{x})$ are feature vectors extracted at edge $k = \{i, j\}$ with labels t_i and t_j on nodes i and j,

respectively. Then we have

$$\mathbf{A}_k = \begin{pmatrix} \boldsymbol{\phi}_k(1,1,\mathbf{x}) & 0 & \dots & 0 \\ 0 & \boldsymbol{\phi}_k(1,2,\mathbf{x}) & 0 & \dots \\ 0 & \dots & 0 & \boldsymbol{\phi}_k(T,T,\mathbf{x}) \end{pmatrix}$$

$$\mathbf{y} = \mathbf{A}_k^{\mathrm{T}}\mathbf{w} \tag{5.31}$$

$$f_k(\mathbf{y}) = \sum_{t_i,t_j} \Psi(y_{t_i,t_j}) q^{\backslash k}(t_i) q^{\backslash k}(t_j) \tag{5.32}$$

where $y_{t_i,t_j} = \mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}_k(t_i,t_j,\mathbf{x})$, and $\Psi(\cdot)$ is the probit function. Clearly, we have $\tilde{q}(\mathbf{w}) = \mathrm{proj}\big[f_k(\mathbf{y})\tilde{q}^{\backslash k}(\mathbf{w})\big]$. Note that $\mathbf{A}_k$ is a T^2 by LT^2 matrix, where L is the length of the feature vector $\boldsymbol{\phi}_k$. And $\mathbf{y}$ is a T^2 by 1 vector, which is shorter than $\mathbf{w}$, whose length is $d = LT^2$, especially when we have a lot of features in $\boldsymbol{\phi}_k$. In other words, the exact term $f_k(\mathbf{y})$ only constrains the distribution $\tilde{q}(\mathbf{w})$ in a smaller subspace. Therefore, it is sensible to use low-rank matrix computation to obtain $\mathbf{m}_w$ and $\mathbf{V}_w$, the mean and the variance of $\tilde{q}(\mathbf{w})$, based on the "leave-one-out" posterior $\tilde{q}^{\backslash k}(\mathbf{w}) \sim \mathcal{N}(\mathbf{m}_w{}^{\backslash k}, \mathbf{V}_w{}^{\backslash k})$.

Appendix A details the derivations of the low-rank matrix computation for obtaining new moments. Here, we simply give the updates:

$$\mathbf{m}_w = \mathbf{m}_w^{\backslash k} + \mathbf{V}_w^{\backslash k}\mathbf{A}_k\mathbf{c} \tag{5.33}$$

$$\mathbf{V}_w = \mathbf{V}_w^{\backslash k} - \mathbf{V}_w^{\backslash k}\mathbf{A}_k\mathbf{D}\mathbf{A}_k^{\mathrm{T}}\mathbf{V}_w^{\backslash k} \tag{5.34}$$

$$\mathbf{c} = (\mathbf{V}_y^{\backslash k})^{-1}\big(\mathbf{m}_y - \mathbf{m}_y^{\backslash k}\big) \tag{5.35}$$

$$\mathbf{D} = (\mathbf{V}_y^{\backslash k})^{-1} - (\mathbf{V}_y^{\backslash k})^{-1}(\mathbf{G}_y - \mathbf{m}_y\mathbf{m}_y^{\mathrm{T}})(\mathbf{V}_y^{\backslash k})^{-1} \tag{5.36}$$

where

$$\mathbf{m}_y = \frac{\int f_k(\mathbf{y})\mathbf{y}\mathcal{N}(\mathbf{y}|\mathbf{m}_y^{\backslash k}, \mathbf{V}_y^{\backslash k})\mathrm{d}\mathbf{y}}{Z} \tag{5.37}$$

$$= \frac{\sum_{t_i,t_j} Z_{t_i,t_j}\mathbf{m}_{y_{t_i,t_j}} q^{\backslash k}(t_i) q^{\backslash k}(t_j)}{Z} \tag{5.38}$$

$$\mathbf{G}_y = \frac{\int f_k(\mathbf{y})\mathbf{y}\mathbf{y}^{\mathrm{T}}\mathcal{N}(\mathbf{y}|\mathbf{m}_y^{\backslash k}, \mathbf{V}_y^{\backslash k})\mathrm{d}\mathbf{y}}{Z} \tag{5.39}$$

$$= \frac{\sum_{t_i,t_j} Z_{t_i,t_j}\mathbf{G}_{y_{t_i,t_j}} q^{\backslash k}(t_i) q^{\backslash k}(t_j)}{Z} \tag{5.40}$$

$$Z = \sum_{t_i,t_j} q^{\backslash k}(t_i,t_j) \int \Psi(y_{t_i,t_j}) \mathcal{N}(\mathbf{y}|\mathbf{m}_y^{\backslash k}, \mathbf{V}_y^{\backslash k}) \mathrm{d}\mathbf{y} \tag{5.41}$$

$$= \sum_{t_i,t_j} q^{\backslash k}(t_i) q^{\backslash k}(t_j) Z_{t_i,t_j} \tag{5.42}$$

where

$$Z_{t_i,t_j} = \int \Psi(y_{i,j}) \mathcal{N}(\mathbf{y}|\mathbf{m}_y^{\backslash k}, \mathbf{V}_y^{\backslash k}) \mathrm{d}\mathbf{y} \tag{5.43}$$

$$= \int \Psi(\mathbf{e}_k^{\mathrm{T}}\mathbf{y}) \mathcal{N}(\mathbf{y}|\mathbf{m}_y^{\backslash k}, \mathbf{V}_y^{\backslash k}) \mathrm{d}\mathbf{y} \tag{5.44}$$

$$= \epsilon + (1-2\epsilon)\Psi(z_{t_i,t_j}) \tag{5.45}$$

$$z_{t_i,t_j} = \frac{\mathbf{e}_k^{\mathrm{T}} \mathbf{m}_y^{\backslash k}}{\sqrt{\mathbf{e}_k^{\mathrm{T}} \mathbf{V}_y^{\backslash k} \mathbf{e}_k + 1}} \tag{5.46}$$

$$\rho_{t_i,t_j} = \frac{1}{\sqrt{\mathbf{e}_k^{\mathrm{T}} \mathbf{V}_y^{\backslash k} \mathbf{e}_k + 1}} \frac{(1 - 2\epsilon \mathcal{N}(z_{t_i,t_j}|0,1))}{\epsilon + (1-2\epsilon)\Psi(z_{t_i,t_j})} \tag{5.47}$$

$$\mathbf{m}_{y_{t_i,t_j}} = \frac{\int \Psi(\mathbf{e}_k^{\mathrm{T}}\mathbf{y}) \mathbf{y} \mathcal{N}(\mathbf{y}|\mathbf{m}_y^{\backslash k}, \mathbf{V}_y^{\backslash k}) \mathrm{d}\mathbf{y}}{Z_{t_i,t_j}} \tag{5.48}$$

$$= \mathbf{m}_y^{\backslash k} + \mathbf{V}_y^{\backslash k} \rho_{t_i,t_j} \mathbf{e}_k \tag{5.49}$$

$$\mathbf{G}_{y_{t_i,t_j}} = \mathbf{V}_y^{\backslash k} - \mathbf{V}_y^{\backslash k} \mathbf{e}_k \big(\frac{\rho_{t_i,t_j}(\mathbf{e}_k^{\mathrm{T}} \mathbf{m}_{y_{t_i,t_j}} + \rho_{t_i,t_j})}{\mathbf{e}_k^{\mathrm{T}} \mathbf{V}_y^{\backslash k} \mathbf{e}_k + 1} \big) \mathbf{e}_k^{\mathrm{T}} \mathbf{V}_y^{\backslash k} \tag{5.50}$$

where $\mathbf{e}_k$ is a vector with all elements being zeros except its k^{th} element being one.

We update $q(t_i)$ and $q(t_j)$ as follows:

$$q(t_i,t_j) = \frac{Z_{t_i,t_j} q^{\backslash k}(t_i) q^{\backslash k}(t_j)}{Z} \tag{5.51}$$

$$q(t_i) = \sum_{t_j} q(t_i,t_j), \quad q(t_j) = \sum_{t_i} q(t_i,t_j) \tag{5.52}$$

Computing new approximation terms

Given the moment matching update, it is easy to compute the new approximation term $\tilde{f}_k(\mathbf{w}) \propto \tilde{q}(\mathbf{w})/\tilde{q}^{\backslash k}(\mathbf{w})$. Since both $\tilde{q}(\mathbf{w})$ and $\tilde{q}^{\backslash k}(\mathbf{w})$ are Gaussians, $\tilde{f}_k(\mathbf{w})$ also has the form of a Gaussian (though its variance may not be positive definite). Suppose $\mathbf{H}_k$ and $\boldsymbol{\tau}_k$ are the natural parameters of the Gaussian $\tilde{f}_k(\mathbf{w})$; $\mathbf{H}_k$ is the inverse of its covariance matrix and $\boldsymbol{\tau}_k$ is the multiplication of the inverse of its covariance matrix and its mean vector. $\mathbf{H}_k$

and $\boldsymbol{\tau}_k$ can be computed as follows:

$$\mathbf{H}_k = (\hat{\mathbf{V}}_w)^{-1} - (\mathbf{V}_w^{\backslash k})^{-1} = \mathbf{A}_k \hat{\mathbf{h}}_k \mathbf{A}_k^{\mathrm{T}} \tag{5.53}$$

$$\hat{\mathbf{h}}_k = \left(\mathbf{D}^{-1} - \mathbf{A}_k^{\mathrm{T}} \mathbf{V}_w^{\backslash k} \mathbf{A}_k\right)^{-1} \tag{5.54}$$

$$\boldsymbol{\tau}_k = (\hat{\mathbf{V}}_w)^{-1} \hat{\mathbf{m}}_w - (\mathbf{V}_w^{\backslash k})^{-1} \mathbf{m}_w^{\backslash k} = \mathbf{A}_k \hat{\boldsymbol{\mu}}_k \tag{5.55}$$

$$\hat{\boldsymbol{\mu}}_k = \mathbf{c} + \hat{\mathbf{h}}_k \mathbf{A}_k^{\mathrm{T}} \hat{\mathbf{m}}_w \tag{5.56}$$

where $\mathbf{c}$ and $\mathbf{D}$ are defined in equations (5.35) and (5.36). Instead of computing $\mathbf{H}_k$ and $\boldsymbol{\tau}_k$, we only need to compute the projected low-rank matrix $\hat{\mathbf{h}}_k$ and the projected vector $\hat{\boldsymbol{\mu}}_k$. This saves both computation time and memory. For example, if the range of $\mathbf{t}$ is $\{1, 2\}$, then A_k is a d by 4 matrix where d is the length of $\mathbf{w}$. Since we often have many features extracted from observations, d tends to be a large number. In this case, $\mathbf{H}_k$ is a huge d by d matrix and $\boldsymbol{\tau}_k$ is a long d by 1 vector, while $\hat{\mathbf{h}}_k$ is only a 4 by 4 matrix and $\hat{\boldsymbol{\tau}}_k$ is a 4 by 1 vector.

To help the algorithm converge, we use a step size λ between 0 and 1 to update the approximation term. Then the damped approximation term $\tilde{f}_k(\mathbf{w})$ has the parameters $\mathbf{h}_k$ and $\boldsymbol{\mu}_k$:

$$\mathbf{h}_k = \lambda \hat{\mathbf{h}}_k + (1 - \lambda) \mathbf{h}_k^{old} \tag{5.57}$$

$$\boldsymbol{\mu}_k = \lambda \hat{\boldsymbol{\mu}}_k + (1 - \lambda) \boldsymbol{\mu}_k^{old} \tag{5.58}$$

where $\mathbf{h}_k^{old}$ and $\boldsymbol{\mu}_k^{old}$ are the old approximation terms for edge k.

Given the damped approximation term $\tilde{f}_k(\mathbf{w})$, we can compute $\tilde{q}(\mathbf{w}) = \tilde{q}^{\backslash k}(\mathbf{w}) \tilde{f}_k(\mathbf{w})$ with mean $\mathbf{m}_w$ and variance $\mathbf{V}_w$ as follows:

$$\mathbf{V}_w = \mathbf{V}_w^{\backslash k} - \mathbf{V}_w^{\backslash k} \mathbf{A}_k (\mathbf{h}_k^{-1} + \mathbf{A}^{\mathrm{T}} \mathbf{V}_w^{\backslash k} \mathbf{A})^{-1} \mathbf{A}_k^{\mathrm{T}} \mathbf{V}_w^{\backslash k} \tag{5.59}$$

$$\mathbf{m}_w = \mathbf{m}_w^{\backslash k} - \mathbf{V}_w^{\backslash k} \mathbf{A}_k (\mathbf{h}_k^{-1} + \mathbf{A}^{\mathrm{T}} \mathbf{V}_w^{\backslash k} \mathbf{A})^{-1} \mathbf{A}_k^{\mathrm{T}} \mathbf{m}_w^{\backslash k} + \mathbf{V}_w \mathbf{A}_k \boldsymbol{\mu}_k \tag{5.60}$$

Also, we compute the approximation terms $\hat{f}_k(t_i)$ and $\hat{f}_k(t_j)$ for the discrete labels as follows:

$$\hat{f}(t_i) = q(t_i) / q^{\backslash k}(t_i) \qquad\qquad \hat{f}(t_j) = q(t_j) / q^{\backslash k}(t_j) \tag{5.61}$$

We use a step size ξ to damp the update for the discrete distribution $q(\mathbf{t})$. Notice that

the approximate distributions of the parameters and the labels, $\tilde{q}(\mathbf{w})$ and $q(\mathbf{t})$, depend on each other. The new $\tilde{q}(\mathbf{w})$ or $q(\mathbf{t})$ relies on both the old $\tilde{q}(\mathbf{w})$ and the old $q(\mathbf{t})$. Especially, a crude approximation $q(\mathbf{t})$ can directly affect the quality of the new mean parameters of $\tilde{q}(\mathbf{w})$. Therefore, we set the step size ξ for $q(\mathbf{t})$ to be smaller than the step size λ for $\tilde{q}(\mathbf{w})$, such that a more refined $\tilde{q}(\mathbf{w})$ is obtained before updating $q(\mathbf{t})$.

Given the step size ξ, the damped approximation terms, $\tilde{f}_k(t_i)$ and $\tilde{f}_k(t_j)$, and the new marginal distributions, $q(t_i)$ and $q(t_j)$, can be computed as follows:

$$\tilde{f}_k(t_i) = \hat{f}(t_i)^{\xi}(\tilde{f}_k(t_i)^{old})^{1-\xi} \qquad \tilde{f}_k(t_j) = \hat{f}(t_j)^{\xi}(\tilde{f}_k(t_j)^{old})^{1-\xi} \tag{5.62}$$

$$q(t_i) = q^{\backslash k}(t_j)\tilde{f}_k(t_i) \qquad q(t_j) = q^{\backslash k}(t_j)\tilde{f}_k(t_j) \tag{5.63}$$

where $\tilde{f}(t_i)_k^{old}$ and $\tilde{f}(t_j)_k^{old}$ are the old approximation terms for edge $k = \{i, j\}$.

Deleting old approximation terms

The parameters of the "leave-one-out" approximate posteriors $\tilde{q}^{\backslash k}(\mathbf{w})$ and $\tilde{q}^{\backslash k}(\mathbf{t})$ can be efficiently computed as follows:

$$\mathbf{V}_w^{\backslash k} = (\mathbf{V}_w^{-1} - \mathbf{A}_k\mathbf{h}_k\mathbf{A}_k^{\mathrm{T}})^{-1} \tag{5.64}$$

$$= \mathbf{V}_w + \mathbf{V}_w\mathbf{A}_k(\mathbf{h}_k^{-1} - \mathbf{A}_k^{\mathrm{T}}\mathbf{V}_w\mathbf{A}_k)^{-1}(\mathbf{V}_w\mathbf{A}_k)^{\mathrm{T}} \tag{5.65}$$

$$\mathbf{m}_w^{\backslash k} = (\mathbf{V}_w^{\backslash k})^{-1}(\mathbf{V}_w^{-1}\mathbf{m}_w - \mathbf{u}_k) \tag{5.66}$$

$$= \mathbf{m}_w + \mathbf{V}_w^{\backslash k}\mathbf{A}_k(\mathbf{h}_k\mathbf{A}_k^{\mathrm{T}}\mathbf{m}_w - \boldsymbol{\mu}_k) \tag{5.67}$$

$$q^{\backslash k}(t_j) = q(t_i)/\tilde{f}_k(t_i) \tag{5.68}$$

$$q^{\backslash k}(t_j) = q(t_j)/\tilde{f}_k(t_j) \tag{5.69}$$

5.3.4 Flattening the approximation structure

In practice, we found that the approximation method, presented in Sections 5.3.1 and 5.3.2, led to a divergence problem in training. In this section, we examine the reason for divergence and propose a method to fix this problem.

The approximation method has two levels of approximations, which are visualized at the top of Figure 5-2. At the upper level of the top graph, the approximation method iteratively refines the approximate posterior $q(\mathbf{w})$ based on the approximation term $\tilde{Z}(\mathbf{w})$;

at the lower level, it iteratively refines $\tilde{Z}(\mathbf{w})$ by individual approximation terms $\{\tilde{f}_k(\mathbf{w})\}$.

A naive implementation of the two-level approximation will always initialize the approximation terms $\{\tilde{f}_k(\mathbf{w})\}$ as 1, such that the iterations at the lower level start from scratch every time. Thus, removing the denominator $\tilde{Z}(\mathbf{w})$ in the upper level amounts to removing all the previous approximation terms $\{\tilde{f}_k(\mathbf{w})\}$ in the lower level. The naive implementation requires the "leave-one-out" approximation $q^{\backslash z}(\mathbf{w}) \propto \frac{q(\mathbf{w})}{\tilde{Z}(\mathbf{w})}$ to have a positive definite covariance matrix. Since this requirement is hard to meet in practice, the training procedure often skips the whole denominator $Z(\mathbf{w})$ in the upper level. This skipping dramatically decreases the approximation accuracy in practice.

A better idea is to initialize the approximation terms using the values obtained from the previous iterations. By doing so, we do not require the covariance of $q^{\backslash z}(\mathbf{w})$ to be positive definite anymore. Instead, we need that of $\tilde{q}^{\backslash k}(\mathbf{w})$ in equation (5.24) to be positive definite, which is easier to satisfy. Now, when the iterations in the lower level start, $\tilde{q}^{\backslash k}(\mathbf{w})$ usually has a positive definite covariance. However, after a few iterations, the covariance of $\tilde{q}^{\backslash k}(\mathbf{w})$ often becomes non-positive definite again. The underlying reason for this instability is that the partition function $Z(\mathbf{w})$ is complicated and difficult to approximate accurately.

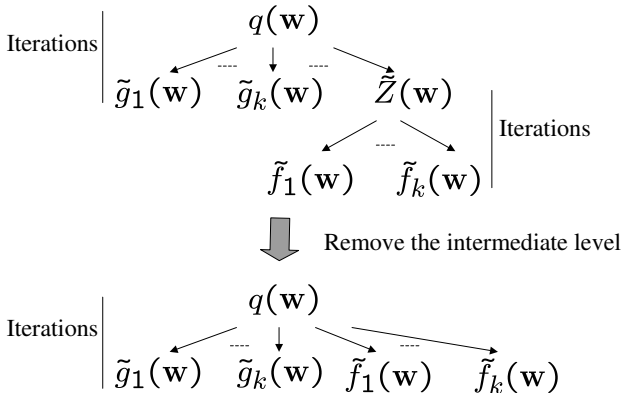


Figure 5-2: Flattening the approximation structure. The upper graph shows the two-level approximation structure of the methods described in the previous sections. The lower graph shows the flattened single-level approximation structure.

To address the problem, we flatten the two-level approximation structure by expanding $\tilde{Z}(\mathbf{w})$ in the upper level. Now we focus on $q(\mathbf{w})$, our ultimate interest in training, without directly dealing with the difficult partition function $Z(\mathbf{w})$. The flattened structure is shown at the bottom of the Figure 5-2. Specifically, the approximate posterior $q(\mathbf{w})$ has the

following form:

$$q(\mathbf{w}) \propto p_0(\mathbf{w}) \prod_{k \in \mathcal{E}} \tilde{g}_k(\mathbf{w}) \frac{1}{\prod_{k \in \mathcal{E}} \tilde{f}_k(\mathbf{w})} \tag{5.70}$$

Equation (5.70) uses the approximation term $\tilde{f}_k(\mathbf{w})$ for each edge rather than using $\tilde{Z}(\mathbf{w})$. It is also possible to interpret the flattened structure from the perspective of the two-level approximation structure. That is, each time we partially update $\tilde{Z}(\mathbf{w})$ based on only one individual term approximation $\tilde{f}_k(\mathbf{w})$, and then refine $q(\mathbf{w})$ before updating $\tilde{Z}(\mathbf{w})$ again based on another individual term approximation.

With the flattened structure, we update and incorporate $\tilde{g}_k(\mathbf{w})$ using standard EP. Also, we have the same updates for $q(\mathbf{t})$ as described in the previous sections. To update and incorporate $\tilde{f}_k(\mathbf{w})$ in the denominator, we can use either power EP or transformed EP for training. Specifically, by inserting $s_k(\mathbf{w}) = f_k(\mathbf{w})$ and $\tilde{s}_k(\mathbf{w}) = \tilde{f}_k(\mathbf{w})$ into equations (5.11) to (5.13) and (5.16) to (5.18) respectively, we obtain the PEP and transformed PEP updates for $\tilde{f}_k(\mathbf{w})$. Setting $m_k = 0$ in transformed PEP gives the transformed EP updates.

Specifically, for PEP, we have the same deletion and moment steps as what we have in Sections 5.3.3 and 5.3.3 to approximate the partition function. We only modify the inclusion step (5.59) and (5.60) as follows:

$$\boldsymbol{\xi}_k = 2\mathbf{h}_k^{old} - \mathbf{h}_k \tag{5.71}$$

$$\boldsymbol{\eta}_k = 2\boldsymbol{\mu}_k^{old} - \boldsymbol{\mu}_k \tag{5.72}$$

$$\mathbf{V}_w = \mathbf{V}_w^{\backslash k} - \mathbf{V}_w^{\backslash k}\mathbf{A}_k(\boldsymbol{\xi}_k^{-1} + \mathbf{A}_k^{\mathrm{T}}\mathbf{V}_w^{\backslash k}\mathbf{A}_k)^{-1}\mathbf{A}_k^{\mathrm{T}}\mathbf{V}_w^{\backslash k} \tag{5.73}$$

$$\mathbf{m}_w = \mathbf{m}_w^{\backslash k} - \mathbf{V}_w^{\backslash k}\mathbf{A}_k(\boldsymbol{\xi}_k^{-1} + \mathbf{A}_k^{\mathrm{T}}\mathbf{V}_w^{\backslash k}\mathbf{A}_k)^{-1}\mathbf{A}_k^{\mathrm{T}}\mathbf{m}_w^{\backslash k} + \mathbf{V}_w\mathbf{A}_k\boldsymbol{\eta}_k \tag{5.74}$$

For transformed EP, we simply flip the sign of $\mathbf{h}_k$ and $\boldsymbol{\mu}_k$ in the deletion and inclusion steps described in Sections 5.3.3 and 5.3.3. The moment matching step remains the same.

Note that both the power EP and transformed EP updates take $O(d^2)$ time, while a simple implementation of the two-level approximation would take $O(d^3)$ time due to the inverse of the covariance matrix of $\tilde{Z}(\mathbf{w})$.

As a result of structure flattening, we have the following advantages over the previous two approaches. First, training can converge easily. Instead of iteratively approximating $Z(\mathbf{w})$ in the lower level based on all the individual terms as before, a partial update based

on only one individual term makes training much more stable. Second, we can obtain a better "leave-one-out" posterior $q^{\backslash k}(\mathbf{w})$, since we update $q(\mathbf{w})$ more frequently than in the previous two cases. A more refined $q(\mathbf{w})$ leads to a better $q^{\backslash k}(\mathbf{w})$, which in turn guides the KL minimization to find a better new $q(\mathbf{w})$. Finally, the flattened approximation structure allows us to process approximation terms in any order. We found empirically that, instead of using a random order, it is better to process the denominator term $\{\tilde{f}_k(\mathbf{w})\}$ directly after processing the numerator term $\{\tilde{g}_k(\mathbf{w})\}$ that is associated with the same edge as $\{\tilde{f}_k(\mathbf{w})\}$. Using this order, the information in a numerator term can guide the approximation for the corresponding denominator term. In return, the denominator term can normalize the approximation obtained from the numerator term immediately, such that $q(\mathbf{w})$ and $q^{\backslash k}(\mathbf{w})$ are more balanced during training. Thus, using this iteration order improves the approximation quality.

Using the flattened structure and pairing the processing of the corresponding numerator and denominator terms, we can train BCRFs robustly. For example, on the tasks of analyzing synthetic datasets in the experimental section, training with the two-level structure breaks down by skipping $\{\tilde{Z}(\mathbf{w})\}$ or $\{\tilde{f}_k(\mathbf{w})\}$ and fails to converge. In contrast, training with the flattened structure converges successfully and leads to a test error around 10% (see the details in the experiment section).

5.4 Inference on BCRFs

Unlike traditional classification problems, where we have a scalar output for each input, a BCRF jointly labels all the hidden vertices in an undirected graph. Given the posterior $q(\mathbf{w})$, we present two ways for inference on BCRFs: Bayesian-point-estimate (BPE) inference and Bayesian model averaging. While the BPE inference uses the posterior mean $\mathbf{m}_w$, as the model parameter vector for inference, Bayesian model averaging integrates over the posterior $q(\mathbf{w})$ instead of using a fixed parameter vector. The advantage of model averaging over the BPE approach is that model averaging makes full use of the data by employing not only the estimated mean of the parameters, but also the estimated uncertainty (variance). The practical benefit of model averaging, which combines predictions across al fits, is the elimination of the overfitting problem. Since exact model averaging is intractable, we propose an efficient way to approximate it. The following sections describe in detail the BPE

inference and approximate model averaging for BCRFs.

5.4.1 BPE inference

Given a new test graph $\mathbf{x}^\star$, a BCRF trained on $(\mathbf{x}, \mathbf{t})$ can approximate the predictive distribution as follows:

$$p(\mathbf{t}^\star|\mathbf{x}^\star, \mathbf{t}, \mathbf{x}) = \int p(\mathbf{t}^\star|\mathbf{x}^\star, \mathbf{w}) p(\mathbf{w}|\mathbf{t}, \mathbf{x}) d\mathbf{w} \tag{5.75}$$

$$\approx p(\mathbf{t}^\star|\mathbf{x}^\star, \mathbf{m}_w) \tag{5.76}$$

$$= \frac{1}{Z(\mathbf{m}_w)} \prod_{\{i,j\}\in\mathcal{E}} g_{i,j}(t_i^\star, t_j^\star, \mathbf{x}; \mathbf{m}_w) \tag{5.77}$$

where $\mathbf{m}_w$ is the mean of the approximate posterior $q(\mathbf{w})$.

Given $\mathbf{m}_w$, we are interested in finding the maximum marginal probability (MMP) configuration of the test graph:

$$t_i^{\text{MMP}} = \text{argmax}_{t_i^\star} P(t_i^\star|\mathbf{x}^\star, \mathbf{m}_w), \quad \forall i \in \mathcal{V}. \tag{5.78}$$

To obtain the MMP configuration, one can use the junction tree algorithm to get the exact marginals of the labels if the tree width of the graph is not very large. If the tree width is relatively large, one can use loopy belief propagation or tree-structured EP (Minka & Qi, 2003) to approximate the marginals. After obtaining the exact or approximate marginals, then one can find the MMP configuration by choosing a label for each node with the maximal marginal.

For applications where a single labeling error on a vertex of a graph can dramatically change the meaning of the whole graph, such as word recognition, we may want to find the maximum joint probability (MJP) configuration. While the maximum marginal configuration will choose the most probable individual labels, the MJP configuration finds a globally compatible label assignment. Depending on applications, we can use MMP or MJP configurations accordingly. In our experiments, we always find MMP configurations.

5.4.2 Approximate model averaging

Given a new test graph $\mathbf{x}^\star$, a BCRF trained on $(\mathbf{x}, \mathbf{t})$ can approximate the predictive distribution as follows:

$$p(\mathbf{t}^\star|\mathbf{x}^\star, \mathbf{t}, \mathbf{x}) = \int p(\mathbf{t}^\star|\mathbf{x}^\star, \mathbf{w}) p(\mathbf{w}|\mathbf{t}, \mathbf{x}) d\mathbf{w} \tag{5.79}$$

$$\approx \int p(\mathbf{t}^\star|\mathbf{x}^\star, \mathbf{w}) q(\mathbf{w}) d\mathbf{w} \tag{5.80}$$

$$= \int \frac{q(\mathbf{w})}{Z(\mathbf{w})} \prod_{\{i,j\}\in\mathcal{E}} g_{i,j}(t_i^\star, t_j^\star, \mathbf{x}^\star; \mathbf{w}) d\mathbf{w} \tag{5.81}$$

where $q(\mathbf{w})$ is the approximation of the true posterior $p(\mathbf{w}|\mathbf{t}, \mathbf{x})$. Since the exact integral for model averaging is intractable, we need to approximate this integral.

We can approximate the predictive posterior term by term as in EP. But typical EP updates will involve the updates over both the parameters and the labels. That is much more expensive than using a point estimate for inference, which involves only updates of the labels. To reduce the computational complexity, we propose the following approach. It is based on the simple fact that without any labels, the test graph does not offer information about the parameters in the conditional model. Mathematically, after integrating out the labels, the posterior distribution $q(\mathbf{w})$ remains the same:

$$q(\mathbf{w})^{new} = \int \frac{1}{Z(\mathbf{w})} \prod_{\{i,j\}\in\mathcal{E}} g_{i,j}(t_i^\star, t_j^\star, \mathbf{x}; \mathbf{w}) q(\mathbf{w}) d\mathbf{t} = q(\mathbf{w}) \tag{5.82}$$

Since $q(\mathbf{w})$ is unchanged, we can update only $q(\mathbf{t}^\star)$ when incorporating one term $g_{i,j}(t_i^\star, t_j^\star, \mathbf{x}; \mathbf{w})$. Specifically, given the posterior $q(\mathbf{w}) \sim \mathcal{N}(\mathbf{m}_w, \mathbf{V}_w)$, we use the factorized approximation $q(\mathbf{t}^\star) = \prod_i q(t_i^\star)$ and update $q(t_i^\star)$ and $q(t_j^\star)$ as follows:

$$z_{t_i^\star, t_j^\star} = \frac{\boldsymbol{\phi}_k^{\mathrm{T}} \mathbf{m}_w^{\backslash k}}{\sqrt{\boldsymbol{\phi}_k^{\mathrm{T}} \mathbf{V}_w^{\backslash k} \boldsymbol{\phi}_k + 1}} \tag{5.83}$$

$$Z_{t_i^\star, t_j^\star} = \epsilon + (1 - 2\epsilon)\Psi(z_{t_i^\star, t_j^\star}) \tag{5.84}$$

$$q(t_i^\star, t_j^\star) = \frac{Z_{t_i^\star, t_j^\star} q^{\backslash k}(t_i^\star) q^{\backslash k}(t_j^\star)}{Z} \tag{5.85}$$

$$q(t_i^\star) = \sum_{t_j} q(t_i^\star, t_j^\star) \tag{5.86}$$

$$q(t_j^\star) = \sum_{t_i} q(t_i^\star, t_j^\star) \tag{5.87}$$

where $\boldsymbol{\phi}_k$ is the feature vector extracted at the k^{th} edge. Note that the deletion and inclusion steps for $q(\mathbf{t}^\star)$ are similar to those described in the previous section on BCRF training.

Compared to the BPE inference, the above algorithm is almost the same except for an importance difference: approximate model averaging uses $\frac{\boldsymbol{\phi}_k^{\mathrm{T}} \mathbf{m}_w^{\backslash k}}{\sqrt{\boldsymbol{\phi}_k^{\mathrm{T}} \mathbf{V}_w^{\backslash k} \boldsymbol{\phi}_k + 1}}$ in its potential function, while the BPE inference simply uses $\boldsymbol{\phi}_k^{\mathrm{T}} \mathbf{m}_w^{\backslash k}$. In other words, the approximate model averaging algorithm normalizes the potentials by the posterior variance projected on feature vectors, while the BPE inference does not.

5.5 Experimental results

This section compares BCRFs with CRFs trained by maximum likelihood (ML) and maximum a posteriori (MAP) methods on several synthetic datasets, a document labeling task, and an ink recognition task, demonstrating BCRFs' superior test performance. ML- and MAP-trained CRFs include CRFs with probit potential functions (5.4) and with exponential potential functions (5.3). We applied both PEP and TEP to train BCRFs. To avoid divergence, we used a small step size. In the following paragraphs, BCRF-PEP and BCRF-TEP denote CRFs trained by PEP and TEP respectively, with model averaging for inference. BPE-CRF-PEP and BPE-CRF-TEP denote variants of BCRFs, i.e., CRFs using the BPE inference. For ML- and MAP-trained CRFs, we applied the junction tree algorithm for inference. We used probit models as the default potential functions by setting $\epsilon = 0$ in equation (5.4), unless we explicitly stated that we used exponential potentials. To compare the test performance of these algorithms, we counted the test errors on all vertices in the test graphs.

5.5.1 Classifying synthetic data

On synthesized datasets, we examined the test performance of BCRFs and MAP-trained probit CRFs for different sizes of training sets and different graphical structures. The labels of the vertices in the synthetic graphs are all binary. The parameter vector $\mathbf{w}$ has 24 elements. The feature vectors $\{\boldsymbol{\phi}_{i,j}\}$ are randomly sampled from one of four Gaussians. We can easily control the discriminability of the data by changing the variance of the

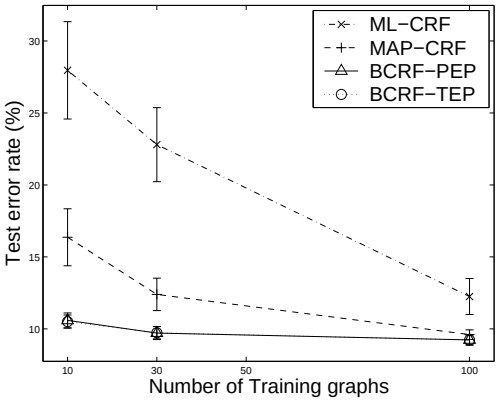


Figure 5-3: Test error rates for ML- and MAP-trained CRFs and BCRFs on synthetic datasets with different numbers of training loops. The results are averaged over 10 runs. Each run has 1000 test loops. Non-overlapping of error bars, the standard errors scaled by 1.64, indicates 95% significance of the performance difference.

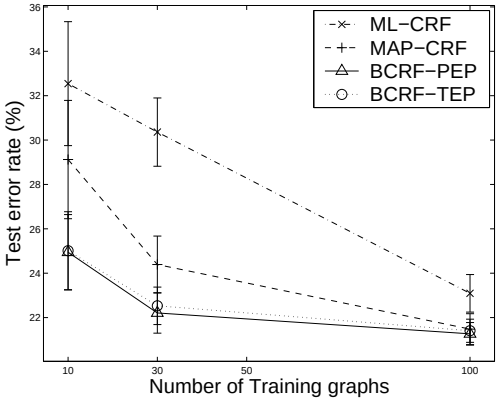


Figure 5-4: Test error rates for ML- and MAP-trained CRFs and BCRFs on synthetic datasets with different numbers of training chains. The results are averaged over 10 runs. Each run has 1000 test chains. Though the error bars overlap a little bit, BCRFs still outperform ML- and MAP-trained CRFs at 95% significance according to t-tests.

Gaussians. Based on the model parameter vector and the sampled feature vectors, we can compute the joint probability of the labels as in equation (5.1) and randomly sample the labels. For BCRFs, we used a step size of 0.8 for training. For MAP-trained CRFs, we used quasi-Newton methods with the BFGS approximation of Hessians (Sha & Pereira, 2003).

Different training sizes for loopy CRFs

Each graph has 3 vertices in a loop. In each trial, 10 loops were sampled for training and 1000 loops for testing. The procedure was repeated for 10 trials. A Gaussian prior with mean $\mathbf{0}$ and diagonal variance $5\mathbf{I}$ was used for both BCRF and MAP CRF training. We repeated the same experiments by increasing the number of training graphs from 10 to 30 to 100.

The results are visualized in Figure 5-3 and summarized in Table 5.1. According to t-tests, which have stronger test power than the error bars in Figure 5-3, both types of BCRFs outperform ML- and MAP-trained CRFs in all cases at 98% statistical significance level. PEP- and TEP-trained BCRFs achieve comparable test accuracy. Furthermore, approximate model averaging improves the test performance over the BPE inference with 86% and 91% statistical significance for the case of 10 and 30 training graphs, respectively. When

Algorithm	Test error rates		
	10 train.	30 train.	100 train.
ML-CRF	27.96 ⋆	22.80 ⋆	12.25 ⋆
MAP-CRF	16.36 ⋆	12.40 ⋆	9.60 ⋆
BPE-CRF-TEP	10.66	9.85	9.21
BPE-CRF-PEP	10.67	9.76	**9.14**
BCRF-TEP	**10.51**	9.73	9.23
BCRF-PEP	10.59	**9.71**	9.23

Table 5.1: Test error rates on synthetic datasets with different numbers of training loops. The results are averaged over 10 runs. Each run has 1000 test loops. The stars indicate that the error rates are worse than those of BCRF-PEP at 98% significance.

more training graphs are available, MAP-trained CRFs and BCRFs perform increasingly similarly, though still statistically differently. The reason is that the posterior is narrower than in the case of fewer training graphs, such that the posterior mode is closer to the posterior mean. In this case, approximate model averaging does not improve test performance, since the inference becomes more sensitive to the error introduced by the approximation in model averaging.

Different training sizes for chain-structured CRFs

We then changed the graphs to be chain-structured. Specifically, each graph has 3 vertices in a chain. In each trial, 10 chains were sampled for training and 1000 chains for testing. The procedure was repeated for 10 trials. A Gaussian prior with mean $\mathbf{0}$ and diagonal variance $5\mathbf{I}$ was used for both BCRF and MAP CRF training. Then we repeated the same experiments by increasing the number of training graphs from 10 to 30 to 100. The results are visualized in Figure 5-4 and summarized in Table 5.2. Again, BCRFs outperform ML- and MAP-trained CRFs with high statistical significance. Also, model averaging significantly improves the test accuracy over the BPE inference, especially when the size of training sets is small.

5.5.2 Labeling FAQs

We compared BCRFs with MAP-trained probit and exponential CRFs on the frequently asked questions (FAQ) dataset, introduced by McCallum et al. (2000). The dataset consists of 47 files, belonging to 7 Usenet newsgroup FAQs. Each file has multiple lines, which can be the header (H), a question (Q), an answer (A), or the tail (T). Since identifying the header

Algorithm	Test error rates		
	10 train.	30 train.	100 train.
ML-CRF	32.54 ⋆	30.36 ⋆	23.09 ⋆
MAP-CRF	29.12 ⋆	24.39 ⋆	21.48
BPE-CRF-TEP	25.74 ⋆	23.06 ⋆	21.42
BPE-CRF-PEP	25.48 ⋆	22.52 ⋆	21.34
BCRF-TEP	25.01	22.53 ⋆	21.41 ⋆
BCRF-PEP	**24.94**	**22.21**	**21.27**

Table 5.2: Test error rates for ML- and MAP-trained CRFs and BCRFs on synthetic datasets with different numbers of training chains. The results are averaged over 10 runs. Each run has 1000 test chains. The stars indicate that the error rates are worse than those of BCRF-PEP at 98% significance.

and the tail is relatively easy, we simplify the task to label only the lines that are questions or answers. To save time, we truncated all the FAQ files such that no file has more than 500 lines. On average, the truncated files have 479 lines. The dataset was randomly split 10 times into 19 training and 28 test files. Each file was modeled by a chain-structured CRF, whose vertices correspond to lines in the files. The feature vector for each edge of a CRF is simply the concatenation of feature vectors extracted at the two neighboring vertices, which are the same set of 24 features, for example, *begins-with-number*, *begins-with-question-word*, and *indented-1-to-4*, as McCallum et al. (2000) used.

The test performance is summarized in Table 5.3 and visualized in Figure 5-5. According to t-tests, BCRFs outperform ML- and MAP-trained CRFs with probit or exponential potentials on the truncated FAQs dataset at 98% statistical significance level.

Finally, we compared PEP- and TEP-trained BCRFs. On one hand, Figures 5-6 and 5-7 show that TEP-trained BCRFs converged steadily, while the change in BCRF parameters in PEP training oscillated. On the other hand, as shown in Figure 5-5 and Table 5.3, PEP-trained BCRFs achieved more accurate test performance than TEP-trained BCRFs, though TEP-trained BCRFs also outperformed MAP-trained CRFs.

5.5.3 Parsing handwritten organization charts

We compared BCRFs and MAP-trained CRFs on ink analysis, a real-world application of computer vision. The goal is to distinguish containers from connectors in drawings of organization charts. A typical organization chart is shown in Figure 5-8. CRFs enable joint classification of all ink on a page. The classification of one ink fragment can influence the

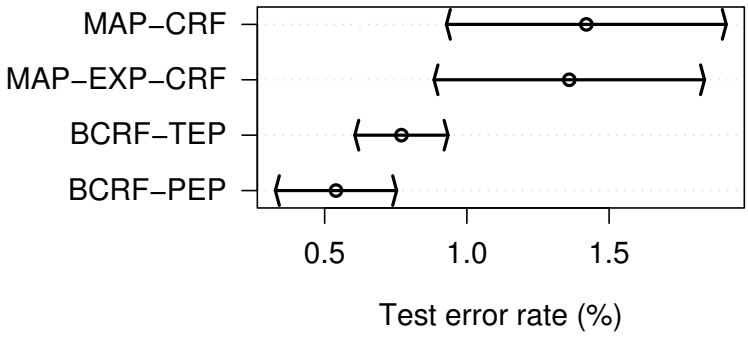


Figure 5-5: Test error rates of different algorithms on FAQ dataset. The results are averaged over 10 random splits. The error bars are the standard errors multiplied by 1.64. Both types of BCRFs outperform MAP-trained CRFs with probit and expontial potentials at 95% statistical significance level, though the eror bar of BCRF-TEP overlaps a little bit with those of MAP-trained CRFs.

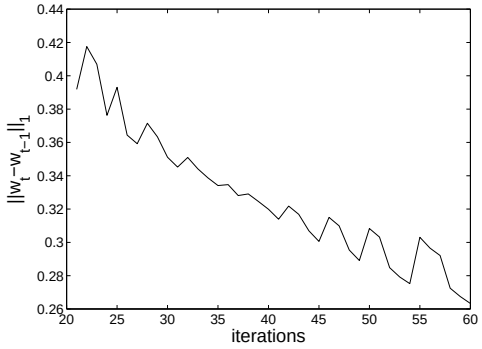


Figure 5-6: Change in parameters along iterations of PEP. Though decreasing overall, the change in parameters oscillates. This slows down the convergence speed.

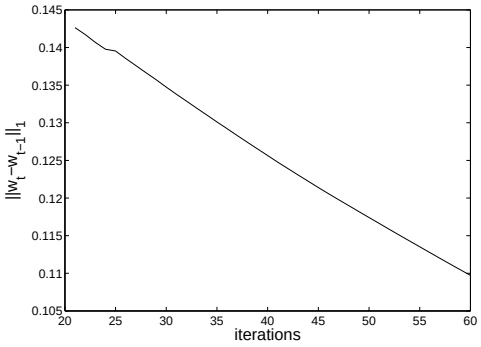


Figure 5-7: Change in parameters along iterations of TEP. The change in parameters monotonically decreases until the training converges.

classification of others, so that context is exploited.

We break the task into three steps:

1. Subdivision of pen strokes into fragments,

2. Construction of a conditional random field on the fragments,

3. Training and inference on the network.

The input is electronic ink recorded as sampled locations of the pen, and collected into strokes separated by pen-down and pen-up events. In the first step, strokes are divided into simpler components called fragments. Fragments should be small enough to belong to a single container or connector. In contrast, strokes can occasionally span more than

Algorithm	Test error rates
MAP-CRF	1.42 ⋆
MAP-Exp-CRF	1.36 ⋆
BPE-CRF-TEP	0.68
BPE-CRF-PEP	0.72 ⋆
BCRF-TEP	0.67
BCRF-PEP	**0.54**

Table 5.3: Test error rates on FAQ dataset. The stars indicate the error rates are worse than that of BCRF-PEP with model averaging at 98% significance. 10 random splits of the dataset were used. Each split has 19 sequences for training and 28 for testing. The average length of the truncated FAQ chains is 479 lines.

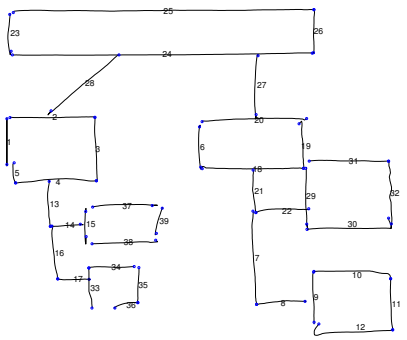


Figure 5-8: An organization chart. Stroke fragments are numbered and their endpoints marked. In the right middle region of the chart, fragments 18, 21, 22, and 29 from both containers and connectors compose a box, which however is not a container. This region is locally ambiguous, but globally unambiguous.

one shape, for example when a user draws a container and a connector without lifting the pen. One could choose the fragments to be individual sampled dots of ink, however, this would be computationally expensive. We choose fragments to be groups of ink dots within a stroke that form straight line segments (within some tolerance) (Figure 5-8).

In the second step, we construct a conditional random field on the fragments. Each ink fragment is represented by a vertex in the graph (Figure 5-9). The vertex has an associated label variable t_i, which takes on the values 1 (container) or 2 (connector). Potential functions quantify the relations between labels given the data. The features used in each potential function include histograms of angles and lengths, as well as templates to detect important shapes, e.g., T-junctions formed by two neighboring fragments.

Finally, we want to train the constructed CRFs and then predict labels of new organization charts based on the trained CRFs.

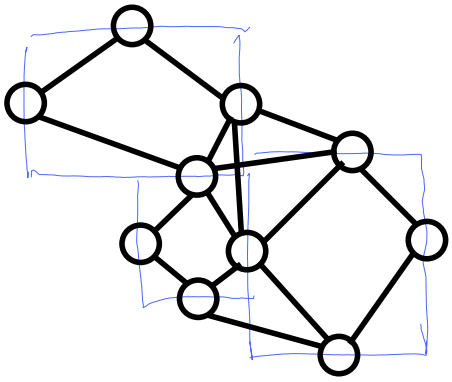


Figure 5-9: A CRF superimposed on part of the chart from Figure 5-8. There is one vertex (shown as circle) per fragment, and edges indicate potentials between neighboring fragments.

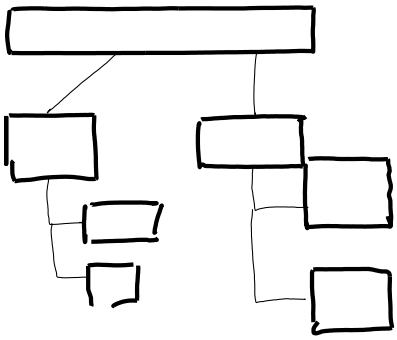


Figure 5-10: Parsing the organization chart by BCRF-PEP. The trained BCRF-PEP labels the containers as bold lines and labels connectors as thin lines. Clearly, by exploiting contextual information in the graph, the BCRF-PEP accurately labels the graph even in the difficult region to the middle-right, where there is local ambiguity.

Algorithm	Test error rates
ML-CRF	10.7 ± 1.21 $\star$
MAP-CRF	6.00 ± 0.64 $\star$
ML-Exp-CRF	10.1 ± 1.21 $\star$
MAP-Exp-CRF	5.20 ± 0.79 $\diamond$
BPE-CRF-PEP	5.93 ± 0.59 $\star$
BCRF-PEP	$\mathbf{4.39 \pm 0.62}$

Table 5.4: Test error rates on organization charts. The results are averaged over 10 random partitions. The stars indicate that the error rates are worse than that of BCRF-TEP at 98% significance. The diamond indicates that BCRF-PEP outperforms MAP-trained CRFs with exponential potentials at 85% significance.

The data set consists of 23 organization charts. We randomly split them 10 times into 14 training and 9 test graphs, and ran BCRFs, ML- and MAP-trained CRFs on each partition.

We used the same spherical Gaussian prior $p(\mathbf{w}) \sim \prod_i \mathcal{N}(w_i|0,5)$ on the parameters $\mathbf{w}$ for both BCRF and MAP CRF training. The same features were also used. Figure 5-10 shows the parsing result of the organization chart in Figure 5-8. The overall test performance is summarized in Table 5.4. Again, BCRFs outperforms the other approaches.

5.6 Conclusions

This chapter has presented BCRFs, a new approach to training and inference on conditional random fields. In training, BCRFs use the TEP method or the PEP method to approximate the posterior distribution of the parameters. Also, BCRFs flatten approximation structures to increase the algorithmic stability, efficiency, and prediction accuracy. In testing, BCRFs approximate model averaging. On synthetic data, FAQ files, and handwritten ink recognition, we compared BCRFs with ML- and MAP-trained CRFs. In almost all the experiments, BCRFs outperformed ML- and MAP-trained CRFs significantly. Also, approximate model averaging enhances the test performance over the BPE inference. Regarding the difference between PEP and TEP training, PEP-trained BCRFs achieve comparable or more accurate test performance than TEP-trained BCRFs, while TEP training tends to have a faster convergence speed than PEP training. Moreover, we need a better theoretical understanding of TEP.

Compared to ML- and MAP-trained CRFs, BCRFs approximate model averaging over the posterior distribution of the parameters, instead of using a MAP or ML point estimate of the parameter vector for inference. Furthermore, BCRF hyperparameters, e.g., the prior variance for the parameters $\mathbf{w}$, can be optimized in a principled way, such as by maximizing model evidence, with parameters integrated out. Similarly, we can use the predictive-ARD-EP method, described in Chapter 4, to do feature selection with BCRFs and to obtain sparse kernelized BCRFs.

Finally, the techniques developed for BCRFs have promise for other machine learning problems. One example is Bayesian learning in Markov random fields. If we set all features $\boldsymbol{\phi}_k$'s of a CRF to be 1, then the CRF will reduce to a Markov random field. Due to this connection, BCRF techniques can be useful for learning Markov random fields. Another example is Bayesian multi-class discrimination. If we have only one edge in a BCRF, then labeling the two neighboring vertices amounts to classifying the edge into multiple classes.

Therefore, we can directly use BCRF techniques for multi-class discrimination by treating each data point as an edge in a CRF with two vertices, which encode the label of the original data point.

Chapter 6

Conclusions and future work

This thesis has presented a series of algorithms to extend expectation propagation (EP) for graphical models, and shown that Bayesian inference and learning can be both efficient and accurate on a variety of real world applications. Specifically, on four types of graphical models, i.e., hybrid Bayesian networks, Markov random fields, independent conditional classification models, and conditional random fields, this thesis extends EP in the following ways:

First, we use window-based EP smoothing as an alternative to EP batch smoothing on dynamic systems, e.g., hybrid Bayesian dynamic networks, for online applications. Window-based EP smoothing finds a trade-off between assumed density filtering (ADF) and batch EP smoothing, and enables EP approximation for online applications. Window-based EP smoothing performs more efficiently than batch EP smoothing by constraining the window length, and more efficiently than sequential Monte Carlo methods by avoiding random samples.

Second, on graphs with loops, we combine a tree-structured EP approximation with the junction tree algorithm, such that we maintain only local consistency of the junction tree representation during updates. This combination greatly improves the computational speed and saves memory. With this combination, an edge in a graph sends messages only to the subtree that is directly connected to the edge, instead of to the whole graph as with a straightforward implementation of structured EP.

Third, we improve classification accuracy in the presence of noisy, irrelevant features. We introduce *predictive* automatic relevance determination (ARD), which chooses the classifier

with the best estimated test performance instead of the one with the maximal evidence. The predictive performance of a classifier is obtained directly from EP updates without carrying out expensive cross validations. Moreover, for training, predictive-ARD does not compute the full covariance matrix of the parameters, but uses sequential updates over features. This greatly improves training efficiency. For testing, predictive-ARD results in sparse models. The model sparsity naturally leads to fast test performance.

Fourth, we apply two extensions of EP, the previously developed power EP (PEP) and the novel transformed PEP methods, to train Bayesian conditional random fields (BCRF). For testing, we average prediction over the estimated posterior distribution of the parameters to avoid overfitting. Moreover, BCRFs achieve efficiency in training and testing by exploring low-ranking matrix operations, which reduces the training time to $O(d^2)$ from $O(d^3)$, and by approximating model averaging respectively. Here d is the dimensionality of the parameters.

The performance of the extended EP algorithms was demonstrated in the following sections:

Section 2.7 A wireless signal detection problem is modeled by hybrid Bayesian networks. On these hybrid Bayesian networks, window-based EP smoothing achieved the bit error rates comparable to those obtained by sequential Monte Carlo methods, i.e., particle smoothers, but with less than one-tenth computational complexity.

Section 3.5 On binary random fields with grid structures and on fully connected binary random fields, the structured EP approximation coupled with the local propagation scheme was more accurate and faster than loopy belief propagation and structured variational methods, as shown in Figures 3-5 and 3-6.

Section 4.4 On gene expression datasets, predictive-ARD-EP achieved more accurate classification results than traditional ARD, which is based on Laplace's approximation and maximizes model evidence, and than support vector machines coupled with feature selection techniques, such as recursive feature elimination and Fisher scoring.

Section 5.5.1 On synthetic chain-structured and on loopy conditional random fields, BCRFs significantly outperformed maximum likelihood (ML) and maximum a posteriori (MAP) trained CRFs in prediction accuracy.

Section 5.5.2 On frequently-asked-question (FAQ) datasets where the goal was to label each line of a FAQ file as a question or an answer, BCRFs significantly lowered the error more than 60% over MAP-trained CRFs.

Section 5.5.3 On ink data where the goal was to recognize containers and connectors in hand-drawn organization charts, BCRFs improved the prediction accuracy over ML- and MAP-trained CRFs with high statistical significance. Moreover, BCRFs provide the basis for Bayesian feature selection to further improve prediction accuracy.

In summary, this thesis has presented extensions of EP that demonstrate the theoretical benefits of Bayesian inference and learning with respect to reducing error, while achieving new practical benefits of efficient computation. Thus, the notion that Bayesian methods are theoretically ideal but computationally impractical is less true now than before this thesis work was done. As to future work, there are many interesting research directions relating to EP and extended EP, including the following:

- Finding a good approximation distribution according to some sensible criteria. For example, in Chapter 3, we used tree-structured approximation distributions, which are able to capture the correlations between nodes in loopy graphs and, at the same time, remain tractable. But finding an optimal tree structure for approximation is a hard problem. In Chapter 3, we used a heuristic that chooses the maximal spanning tree where the weight for each edge is the estimated mutual information between the two neighboring nodes. We estimated the mutual information using only the potential function involving these two nodes. The maximum spanning tree would be optimal according to the maximum likelihood criterion, if we have the true mutual information for each edge, instead of the estimated mutual information (Chow & Liu, 1968). For continuous variables, it may be possible to use approximation distributions other than Gaussian distributions, so that we can model the skewness or the multimodality of some probability distributions. However, the use of other approximation distributions may make it difficult to apply EP directly and may necessitate extensions of EP.

- Combining EP with other approximation methods, especially randomized methods, to take advantage of the benefits of both randomized and deterministic methods. Sudderth et al. (2003) have developed nonparametric belief propagation (BP), where random samples approximate messages used in belief propagation. Since EP is an

extension of BP, a natural generalization of this work would be to combine sampling methods with expectation propagation.

- Estimating approximation and prediction error. To deepen the understanding of EP as well as its extensions, it is important to obtain error bounds for EP estimation and prediction, especially error bounds that are algorithm- and data-dependent. To this end, we can resort to statistical analysis tools in computational learning theory, including PAC-Bayesian theory and its extensions, variational Chernoff bounds, and covering number bounds (Meir & Zhang, 2003; Ravikumar & Lafferty, 2004; Herbrich, 2002; Tong, 2002).

- Exploring the relation of EP to optimization. EP is strongly linked to optimization techniques. For example, EP is similar to the working set method for optimization: both update the factors in the working set, while holding the other factors fixed, and iterate the process until the convergence. The link between EP and optimization methods implies that it is possible to adopt techniques that are well developed in the optimization community to extend EP for efficiency and other purposes. In return, we can modify EP for optimization purposes, by simply replacing moment matching with mode matching. One example is Laplace propagation (Smola et al., 2004), which modifies EP to approximate the mode of Gaussian processes. EP could be similarly modified so as to approximate the most probable configuration for a discrete Markov random field.

- Exploring information geometry to obtain insight into EP and its extensions. Ikeda et al. (2004) have applied information geometry to analyze and improve BP. A possible generalization of this work would be to apply information geometry to analyze EP and its extension, gaining insight into EP from a new perspective and leading to improvements in performance.

- Applying Bayesian techniques to more real world applications. The applications presented in this thesis are only a small subset of many real-world applications, to which extended EP can apply, such as functional genomics, protein structure analysis, digital wireless communications, and information retrieval. Furthermore, because of the constant improvement in Bayesian approximation techniques and the strong modeling

power of graphical models, one can expect that Bayesian inference and learning on graphical models will play an increasingly more influential role in engineering and scientific research.

Appendix A

Low-rank matrix computation for first and second order moments

Define $\mathbf{y} = \mathbf{A}_k^{\mathrm{T}}\mathbf{w}, \mathbf{m}_y^{\backslash k} = \mathbf{A}_k^{\mathrm{T}}\mathbf{m}_w^{\backslash k}$, and $\mathbf{V}_y = \mathbf{A}_k^{\mathrm{T}}\mathbf{V}_w^{\backslash k}\mathbf{A}_k$. It follows that

$$\nabla_m \log Z(\mathbf{m}_w^{\backslash k}, \mathbf{V}_w^{\backslash k}) = \nabla_m \log \int f(\mathbf{A}_k^{\mathrm{T}}\mathbf{w})\mathcal{N}(\mathbf{w}|\mathbf{m}_w^{\backslash k}, \mathbf{V}_w^{\backslash k})\mathrm{d}\mathbf{w} \tag{A.1}$$

$$= \frac{1}{Z}\int \nabla_m f(\mathbf{A}_k^{\mathrm{T}}\mathbf{w})\mathcal{N}(\mathbf{w}|\mathbf{m}_w^{\backslash k}, \mathbf{V}_w^{\backslash k})\mathrm{d}\mathbf{w} \tag{A.2}$$

$$= \frac{1}{Z}\int f(\mathbf{y})\nabla_m \mathcal{N}(\mathbf{y}|\mathbf{m}_y^{\backslash k}, \mathbf{V}_y^{\backslash k})\mathrm{d}\mathbf{y} \tag{A.3}$$

$$= \frac{1}{Z}\int f(\mathbf{y})(\mathbf{y} - \mathbf{m}_y^{\backslash k})^{\mathrm{T}}(\mathbf{V}_y^{\backslash k})^{-1}\mathcal{N}(\mathbf{y}|\mathbf{m}_y^{\backslash k}, \mathbf{V}_y^{\backslash k})\nabla_m \mathbf{m}_y^{\backslash k}\mathrm{d}\mathbf{y} \tag{A.4}$$

$$= \frac{1}{Z}\int f(\mathbf{y})(\mathbf{y} - \mathbf{m}_y^{\backslash k})^{\mathrm{T}}\mathcal{N}(\mathbf{y}|\mathbf{m}_y^{\backslash k}, \mathbf{V}_y^{\backslash k})(\mathbf{V}_y^{\backslash k})^{-1}\mathbf{A}_k^{\mathrm{T}}\mathrm{d}\mathbf{y} \tag{A.5}$$

$$= \mathbf{c}^{\mathrm{T}}\mathbf{A}_k^{\mathrm{T}} \tag{A.6}$$

where $\mathbf{c} = (\mathbf{V}_y^{\backslash k})^{-1}(\mathbf{m}_y - \mathbf{m}_y^{\backslash k})$, and $\mathbf{m}_y = \frac{\int f(\mathbf{y})\mathbf{y}\mathcal{N}(\mathbf{y}|\mathbf{m}_y^{\backslash k}, \mathbf{V}_y^{\backslash k})\mathrm{d}\mathbf{y}}{Z}$ is the new mean of $\mathbf{y}$. Also, we have

$$\nabla_v \log Z(\mathbf{m}_w^{\backslash k}, \mathbf{V}_w^{\backslash k}) = \frac{1}{Z}\int f(\mathbf{y})\nabla_v \mathcal{N}(\mathbf{y}|\mathbf{m}_y^{\backslash k}, \mathbf{V}_y^{\backslash k})\mathrm{d}\mathbf{y} \tag{A.7}$$

$$= -\frac{1}{2}\mathbf{A}_k \frac{1}{Z}\int f(\mathbf{y})\mathcal{N}(\mathbf{y}|\mathbf{m}_y^{\backslash k}, \mathbf{V}_y^{\backslash k})\big((\mathbf{V}_y^{\backslash k})^{-1} -$$

$$(\mathbf{V}_y^{\backslash k})^{-1}(\mathbf{y} - \mathbf{m}_y^{\backslash k})(\mathbf{y} - \mathbf{m}_y^{\backslash k})^{\mathrm{T}}(\mathbf{V}_y^{\backslash k})^{-1}\big)\mathbf{A}_k^{\mathrm{T}}\mathrm{d}\mathbf{y} \tag{A.8}$$

$$= -\frac{1}{2}\mathbf{A}_k\big((\mathbf{V}_y^{\backslash k})^{-1} - (\mathbf{V}_y^{\backslash k})^{-1}(\mathbf{G}_y - 2\mathbf{m}_y(\mathbf{m}_y^{\backslash k})^{\mathrm{T}} +$$

$$\mathbf{m}_y^{\backslash k}(\mathbf{m}_y^{\backslash k})^{\mathrm{T}})(\mathbf{V}_y^{\backslash k})^{-1}\big)\mathbf{A}_k^{\mathrm{T}} \tag{A.9}$$

where $\mathbf{G}_y = \frac{\int f(\mathbf{y})\mathbf{y}\mathbf{y}^{\mathrm{T}}\mathcal{N}(\mathbf{y}|\mathbf{m}_y^{\backslash k}, \mathbf{V}_y^{\backslash k})\mathrm{d}\mathbf{y}}{Z}$ is the new variance of $\mathbf{y}$.

From (A.6) and (A.9), it follows that

$$(\nabla_m \nabla_m^{\mathrm{T}} - 2\nabla_v) \log Z(\mathbf{m}_w^{\backslash k}, \mathbf{V}_w^{\backslash k}) = \mathbf{A}_k \mathbf{D} \mathbf{A}_k^{\mathrm{T}} \tag{A.10}$$

where

$$\mathbf{D} = (\mathbf{V}_y^{\backslash k})^{-1} - (\mathbf{V}_y^{\backslash k})^{-1}(\mathbf{G}_y - \mathbf{m}_y \mathbf{m}_y^{\mathrm{T}})(\mathbf{V}_y^{\backslash k})^{-1} \tag{A.11}$$

The new mean and variance of the parameters $\mathbf{w}$ are

$$\mathbf{m}_w = \mathbf{m}_w^{\backslash k} + \mathbf{V}_w^{\backslash k} \nabla_m^{\mathrm{T}} \log Z(\mathbf{m}_w^{\backslash k}, \mathbf{V}_w^{\backslash k})\big) \tag{A.12}$$

$$= \mathbf{m}_w^{\backslash k} + \mathbf{V}_w^{\backslash k} \mathbf{A}_k \mathbf{c} \tag{A.13}$$

$$\mathbf{V}_w = \mathbf{V}_w^{\backslash k} - \mathbf{V}_w^{\backslash k} (\nabla_m \nabla_m^{\mathrm{T}} - 2\nabla_v) \log Z(\mathbf{m}_w^{\backslash k}, \mathbf{V}_w^{\backslash k}) \mathbf{V}_w^{\backslash k} \tag{A.14}$$

$$= \mathbf{V}_w^{\backslash k} - \mathbf{V}_w^{\backslash k} \mathbf{A}_k \mathbf{D} \mathbf{A}_k^{\mathrm{T}} \mathbf{V}_w^{\backslash k} \tag{A.15}$$

Bibliography

Beal, M. (2003). *Variational algorithms for approximate Bayesian inference*. Doctoral dissertation, Gatsby Computational Neuroscience Unit, University College London.

Bishop, C. (1995). *Neural networks for pattern recognition*. Oxford University Press.

Boyen, X., & Koller, D. (1998). Tractable inference for complex stochastic processes. *Uncertainty in AI*.

Cargnoni, C., Mueller, P., & West, M. (1997). Bayesian forecasting of multinomial time series through conditionally Gaussian dynamic models. *Journal of American Statistical Association*, *92*, 587–606. `ftp://ftp.stat.duke.edu/pub/WorkingPapers/95-22.ps`.

Chen, R., Wang, X., & Liu, J. S. (2000). Adaptive joint detection and decoding in flat-fading channels via mixture Kalman filtering. *IEEE Transactions on Information Theory*, *46*, 2079–2094.

Chow, C. K., & Liu, C. N. (1968). Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, *14*, 462–467.

Doucet, A., de Freitas, N., & Gordon, N. (Eds.). (2001). *Sequential Monte Carlo methods in practice*. Springer-Verlag.

Faul, A. C., & Tipping, M. E. (2002). Analysis of sparse Bayesian learning. *Advances in Neural Information Processing Systems 14* (pp. 383–389).

Fong, W., Godsill, S., Doucet, A., & West, M. (2001). Monte Carlo smoothing with application to audio signal enhancement. *IEEE Trans. on Signal Processing, to appear*. `http://citeseer.nj.nec.com/436555.html`.

Frey, B. J., Patrascu, R., Jaakkola, T., & Moran, J. (2000). Sequentially fitting inclusive trees for inference in noisy-OR networks. *NIPS 13*.

Ghahramani, Z., & Beal, M. J. (2000). Propagation algorithms for variational Bayesian learning. *NIPS 13*. Cambridge, MA: MIT Press.

Ghahramani, Z., & Jordan, M. I. (1997). Factorial hidden Markov models. *Machine Learning*, *29*, 245–273.

Guyon, I., & Elisseeff, A. (2003). Special issue on variable and feature selection. *Journal of Machine Learning Research*. http://www.jmlr.org/papers/special/feature.html.

Herbrich, R. (2002). *Learning kernel classifiers: Theory and algorithms*. MIT Press.

Herbrich, R., Graepel, T., & Campbell, C. (1999). Bayes point machine: Estimating the Bayes point in kernel space. *IJCAI Workshop SVMs* (pp. 23–27).

Heskes, T., & Zoeter, O. (2002). Expectation propagation for approximate inference in dynamic Bayesian networks. *Proc UAI*.

Ikeda, S., Tanaka, T., & Amari, S. (2004). Stochastic reasoning, free energy, and information geometry. *Neural Computation*, *16*, 1779–1810.

Jordan, M. I., Ghahramani, Z., Jaakkola, T. S., & Saul, L. K. (1998). An introduction to variational methods in graphical models. *Learning in Graphical Models*. http://www.ai.mit.edu/~tommi/papers.html.

K. P. Murphy, Y. Weiss, M. I. J. (1999). Loopy belief propagation for approximate inference: An empirical study. *Proc of UAI*.

Kappen, H. J., & Wiegerinck, W. (2001). Novel iteration schemes for the cluster variation method. *NIPS 14*.

Kumar, S., & Hebert, M. (2004). Discriminative fields for modeling spatial dependencies in natural images. *Advances in Neural Information Processing Systems 16*.

Kushner, H. J., & Budhiraja, A. S. (2000). A nonlinear filtering algorithm based on an approximation of the conditional distribution. *IEEE Transaction Automatic Control*, *45*, 580–585.

Lafferty, J., McCallum, A., & Pereira, F. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. *Proc. 18th International Conf. on Machine Learning* (pp. 282–289). Morgan Kaufmann, San Francisco, CA.

Lafferty, J., Zhu, X., & Liu, Y. (2004). Kernel conditional random fields: Representation and clique selection. *Proc. International Conf. on Machine Learning.*

Li, Y., Campbell, C., & Tipping, M. E. (2002). Bayesian automatic relevance determination algorithms for classifying gene expression data. *Bioinformatics*, *18*, 1332–1339.

Liu, J. S. (2001). *Monte Carlo strategies in scientific computing.* Springer-Verlag.

MacKay, D. J. (1992). Bayesian interpolation. *Neural Computation*, *4*, 415–447.

MacKay, D. J. (1998). Introduction to Monte Carlo methods. *Learning in Graphical Models.* http://wol.ra.phy.cam.ac.uk/mackay/BayesMC.html.

Madsen, A. L., & Jensen, F. V. (1998). Lazy propagation in junction trees. *Proc UAI.*

Maybeck, P. S. (1979). *Stochastic models, estimation, and control*, vol. 141 of *Mathematics in Science and Engineering.* Academic Press.

McCallum, A. (2003). Efficiently inducing features of conditional random fields. *Proceedings of the 16th Conference in Uncertainty in Artificial Intelligence.*

McCallum, A., Freitag, D., & Pereira, F. (2000). Maximum entropy Markov models for information extraction and segmentation. *Proc. 17th International Conf. on Machine Learning* (pp. 591–598). Morgan Kaufmann, San Francisco, CA.

Meir, R., & Zhang, T. (2003). Generalization error bounds for Bayesian mixture algorithms. *Journal of Machine Learning Research*, *4*, 839–860.

Minka, T., & Qi, Y. (2003). Tree-structured approximations by expectation propagation. *Advances in Neural Information Processing System 16.* MIT Press.

Minka, T. P. (1998). *From hidden Markov models to linear dynamical systems* (Technical Report 531). Vision and Modeling Group of Media Lab, MIT.

Minka, T. P. (2001). Expectation propagation for approximate Bayesian inference. *Uncertainty in AI'01.* `http://www.stat.cmu.edu/~minka/papers/ep/`.

Minka, T. P. (2004). Power EP. `http://www.stat.cmu.edu/~minka/`.

Minka, T. P., & Lafferty, J. (2002). Expectation propagation for the generative aspect model. *Proc UAI*.

Murphy, K. (2001). The Bayes Net Toolbox for Matlab. *Computing Science and Statistics*, *33*.

Murray, I., & Ghahramani, Z. (2004). Bayesian learning in undirected graphical models: Approximate MCMC algorithms. *UAI*.

Neal, R. M. (1996). *BAyesian learning for neural networks*. No. 118 in Lecture Notes in Statistics. New York: Springer.

Opper, M., & Winther, O. (2000). Gaussian processes for classification: Mean field algorithms. *Neural Computation*.

Pearl, J. (1988). *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann.

Qi, Y., & Minka, T. P. (2003). Expectation propagation for signal detection in flat-fading channels. *Proceedings of IEEE International Symposium on Information Theory*.

Qi, Y., Minka, T. P., Picard, R. W., & Ghahramani, Z. (2004). Predictive automatic relevance determination by expectation propagation. *Proceedings of International Conference on Machine Learning*.

Rätsch, G., Onoda, T., & Müller, K.-R. (2001). Soft margins for AdaBoost. *Machine Learning*, *42*, 287–320.

Ravikumar, P., & Lafferty, J. (2004). Variational Chernoff bounds for graphical models. *Uncertainty in Artificial Intelligence (UAI)*.

Sha, F., & Pereira, F. (2003). *Parsing with conditional random fields* (Technical Report MS-CIS-02-35). University of Pennsylvania.

Smola, A., Vishwanathan, V., & Eskin, E. (2004). Laplace propagation. In *Advances in neural information processing systems 16*. Cambridge, MA: MIT Press.

Sudderth, E., Ihler, A., Freeman, W., & Willsky, A. (2003). Nonparametric belief propagation. *CVPR*.

Tanizaki, H. (2000). Nonlinear and non-Gaussian state-space modeling with Monte Carlo techniques: A survey and comparative study. In C. Rao and D. Shanbhag (Eds.), *Handbook of statistics: Stochastic processes: Modeling and simulation*, chapter 22. North-Holland.

Taskar, B., Guestrin, C., & Koller, D. (2004). Max-margin Markov networks. In S. Thrun, L. Saul and B. Schölkopf (Eds.), *Advances in neural information processing systems 16*.

Teh, Y. W., & Welling, M. (2001). The unified propagation and scaling algorithm. *NIPS 14*.

Tipping, M. E. (2000). The relevance vector machine. *NIPS* (pp. 652–658). The MIT Press.

Tong, Z. (2002). Covering number bounds of certain regularized linear function classes. *The Journal of Machine Learning Research*, *2*, 527–550.

Wainwright, M. J., Jaakkola, T. S., & Willsky, A. S. (2001). Tree-based reparameterization for approximate estimation on loopy graphs. *NIPS 14*.

Wainwright, M. J., Jaakkola, T. S., & Willsky, A. S. (2002). A new class of upper bounds on the log partition function. *Proc UAI*.

Wan, E. A., van der Merwe, R., & Nelson, A. T. (2000). Dual estimation and the unscented transformation. *NIPS 12* (pp. 666–672). `http://cslu.ece.ogi.edu/nsel/research/ukf.html`.

Wang, X., Chen, R., & Guo, D. (2002). Delayed-pilot sampling for mixture Kalman filter with application in fading channels. *IEEE Transactions on Signal Processing, Special Issue on Monte Carlo Methods for Statistical Signal Processing,*, *50*, 241–254.

Welch, G., & Bishop, G. (2002). *An introduction to the Kalman filter* (Technical Report). Department of Computer Science and Engineering, University of North Carolina at Chapel Hill.

Welling, M., & Teh, Y. W. (2001). Belief optimization for binary networks: A stable alternative to loopy belief propagation. *Proc UAI*.

Wiegerinck, W. (2000). Variational approximations between mean field theory and the junction tree algorithm. *Proc UAI*.

Wiegerinck, W., & Heskes, T. (2002). Fractional belief propagation. *NIPS 15*.

Yedidia, J., Freeman, W., & Weiss, Y. (2002). *Constructing free energy approximations and generalized belief propagation algorithms* (Technical Report 2002-40). MERL Research Lab.

Yedidia, J. S., Freeman, W. T., & Weiss, Y. (2000). Generalized belief propagation. *NIPS 13*.

Yuille, A. (2002). A double-loop algorithm to minimize the Bethe and Kikuchi free energies. *Neural Computation*.

Zwitter, M., & Soklic, M. (1998). This breast cancer domain was obtained from the University Medical Centre, Institute of Oncology, Ljubljana, Yugoslavia.