Abstract

Title of dissertation: TRACTABLE LEARNING AND INFERENCE IN HIGH-TREEWIDTH GRAPHICAL MODELS

Justin Domke, Doctor of Philosophy, 2009

Dissertation directed by: Professor Yiannis Aloimonos
Department of Computer Science

Probabilistic graphical models, by making conditional independence assumptions, can represent complex joint distributions in a factorized form. However, in large problems graphical models often run into two issues. First, in non-treelike graphs, computational issues frustrate exact inference. There are several approximate inference algorithms that, while often working well, do not obey approximation bounds. Second, traditional learning methods are non-robust with respect to model errors– if the conditional independence assumptions of the model are violated, poor predictions can result.

This thesis proposes two new methods for learning parameters of graphical models: implicit and procedural fitting. The goal of these methods is to improve the results of running a particular inference algorithm. Implicit fitting views inference as a large nonlinear energy function over predicted marginals. During learning, the parameters are adjusted to place the minima of this function close to the true marginals. Inspired by algorithms like loopy belief propagation, procedural fitting considers inference as a message passing procedure. Parameters are adjusted while learning so that this message-passing process gives the best results. These methods are robust to both

model errors and approximate inference because learning is done directly in terms of predictive accuracy.

Tractable Learning and Inference in High-Treewidth Graphical Models

Justin Domke

Dissertation submitted to the Faculty of the Graduate School of the University of Maryland, College Park in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

2009

Committee:

Professor Yiannis Aloimonos, Chair

Dr. Cornelia Fermüller

Professor David Jacobs

Professor P. S. Krishnaprasad, Dean's representative

Professor Ben Taskar

Contents

List of Figures

List of Tables

Chapter 1

Overview

This overview tries to give a high-level "tour" of the rest of the thesis. The goal is to informally convey the main results and ideas, with a minimum of technical details.

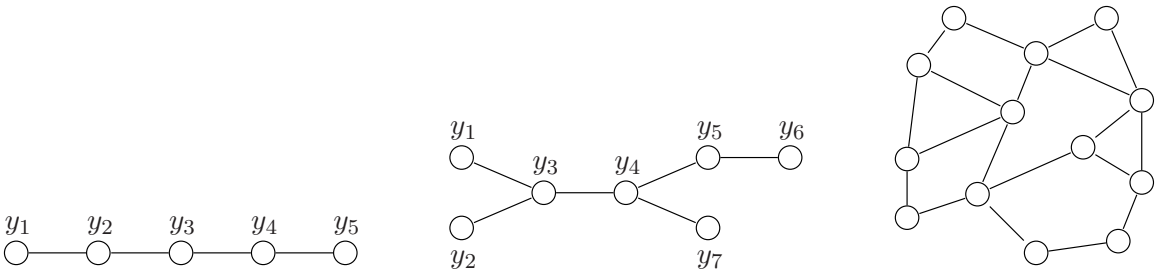
Chapter 2: Background on Graphical Models

Graphical Models. A graphical model is a probability distribution over a set of variables, written in a factorized form. The probability of a configuration is proportional to the product of "factors", each defined on subsets of variables. There are several types of graphical models, but the focus here is on "Conditional Random Fields" (CRFs). These represent the conditional probability of some vector $\mathbf{y}$ given an observation $\mathbf{x}$ as

$$p(\mathbf{y}|\mathbf{x}) \propto \prod_c \psi(\mathbf{y}_c, \mathbf{x}). \tag{1.1}$$

The factors ψ do not have an immediate probabilistic interpretation. Some clarification of notation:

- Each c denotes a subset of variables. A subscript of a set indicates the vector of values in in that set. For example, if $c = \{1, 5, 7\}$, then $\mathbf{y}_c = (y_1, y_5, y_7)$.



(a) A chain graph. (b) A Tree graph. (c) A general graph.

Figure 1.1: Example CRFs

- Each subset of variables c corresponds to a different function ψ. It would be more correct to write the factors as $\psi_c(\mathbf{y}_c, \mathbf{x})$, but the repeated subscript of c becomes tedious.

It is convenient to picture graphical models by drawing a graph with one node for each variable y_i. A pair (i, j) will have an edge if there is a c such that $i \in c$ and $j \in c$. (Since each factor can depend on $\mathbf{x}$ arbitrarily, it is often better to ignore $\mathbf{x}$ when drawing the graph.) Three example CRFs are pictured in Fig. 1.1. The chain-structured graph represents a distribution of the form

$$p(\mathbf{y}|\mathbf{x}) \propto \psi(y_1, y_2, \mathbf{x})\psi(y_2, y_3, \mathbf{x})\psi(y_3, y_4, \mathbf{x})\psi(y_4, y_5, \mathbf{x}),$$

while the tree-structured graph represents a distribution of the form

$$p(\mathbf{y}|\mathbf{x}) \propto \psi(y_1, y_3, \mathbf{x})\psi(y_2, y_3, \mathbf{x})\psi(y_3, y_4, \mathbf{x})\psi(y_4, y_5, \mathbf{x})\psi(y_5, y_6, \mathbf{x})\psi(y_4, y_7, \mathbf{x}).$$

Two Basic Problems. There are two major problems to be solved with graphical models: learning and inference. In learning, one has training data generated by an unknown distribution. The problem is to adjust the parameters of the model to best represent that true distribution. The meaning of "best", of course, must be made precise. The focus of this thesis is learning. However, the driving philosophy is that

the best way to learn depends on how the model will be used.

In inference, one asks questions of the distribution. There are many questions, but our focus here is the problem of *marginalization*, where one seeks the (marginal) probability of subsets of variables, independent of others. For example, one might like univariate marginals like

$$p(y_i^*|\mathbf{x}) = \sum_{\mathbf{y}:y_i=y_i^*} p(\mathbf{y}|\mathbf{x}). \tag{1.2}$$

Note that the number of terms in this sum is exponential in the dimensionality of $\mathbf{y}$. Thus, unless there are few variables, marginals cannot be tractably computed by a direct sum as in Eq. 1.2.

It turns out that, for chain graphs like Fig. 1.1 (a), the summation can be done efficiently by dynamic programming. This algorithm can be rephrased into a message-passing form, where "messages" are sent between neighboring variables. This algorithm, known as Belief-Propagation, is identical to dynamic programming for chains, but also allows for exact and efficient inference in treelike graphs, such as Fig. 1.1 (b).

Computational Intractability. What about densely connected graphs, like Fig. 1.1 (c)? One strategy, if the graph is nearly a tree, is to create an equivalent treelike graph by substituting "super-variables" with one value for each joint configuration of several variables in the original graph. The Junction Tree algorithm, based on this idea, can always compute exact marginals. But it is exponential in the treewidth, or the number of variables that must be aggregated into a single super-variable.

In general graphs, even approximate inference is known to be NP-hard [1, 2]. Still, there are heuristic procedures that often work well. The most popular, Loopy Belief-Propagation (LBP) is essentially just running the Belief-Propagation algorithm, despite the fact that the graph is not a tree. The difference is that one must initialize messages somehow, then iterate until they– hopefully– converge.

Marginal Inference as an Optimization. One can gain understanding into what LBP is actually doing by looking at it from an optimization perspective. One can create an energy function[1] that, when minimized under certain constraints, yields the true marginals. In treelike graphs, this function can be minimized exactly. In general graphs, LBP can be seen as trying to minimize an *approximate* energy function under *approximate* constraints. Let "beliefs" be a shorthand for "predicted marginals". Specifically, LBP tries to minimize the Bethe approximation

$$\sum_c \sum_{\mathbf{x}_c} b(\mathbf{x}_c) \log b(\mathbf{x}_c) + \sum_i n_i \sum_{x_i} b(x_i) \log b(x_i) - \sum_c \sum_{\mathbf{x}_c} b(\mathbf{x}_c) \log \psi(\mathbf{x}_c), \qquad (1.3)$$

over the beliefs b under *local consistency* constraints enforcing, essentially, that univariate and clique-wise beliefs form valid (positive, normalized) probability distributions

$$\begin{aligned} b(x_i) \geq 0 \qquad & b(\mathbf{x}_c) \geq 0 \\ \sum_{x_i} b(x_i) = 1 \qquad & \sum_{\mathbf{x}_c} b(\mathbf{x}_c) = 1, \end{aligned} \qquad (1.4)$$

and that clique beliefs agree with univariate beliefs, i.e.

$$b(x_i) = \sum_{\mathbf{x_c} \setminus i} b(\mathbf{x}_c). \qquad (1.5)$$

Here, $n_i = 1 - |\{c : i \in c\}|$ is a constant for each variable, determined by the number of connected factors.

In general graphs, this function is non-convex, and so LBP can converge to local

[1]The terminology of an "energy" function will be used to contrast between learning and inference: Inference objectives will be called "energy" functions, while learning objectives will be called "loss" functions.

minima. In fact, LBP is not guaranteed to converge at all, although there are various heuristics (such as "damping" of updates) that work well in practice.

Chapter 3: Loss Functions

Essentially, this thesis is about loss functions. Given a model, and some training data, a "loss" measures the quality of fit. Learning consists of adjusting the model to optimize the loss. Why are new loss functions needed? There are two basic reasons: model misspecification, and intractable inference.

Model Misspecification. The first issue, also known as "model error" or "systematic error", is discussed in Chapter 3. Whenever one specifies a graphical model, one makes assumptions. First, the structure of the graph represents assumptions of conditional independence. Second, in most real applications, one selects "features", or restricts factors in the graph to a parametric form. Because of all this, the graphical model cannot represent any probability distribution, but only a restricted set P. Now, let p_0 be the true data-generating distribution. The model is said to be well-specified if $p_0 \in P$.

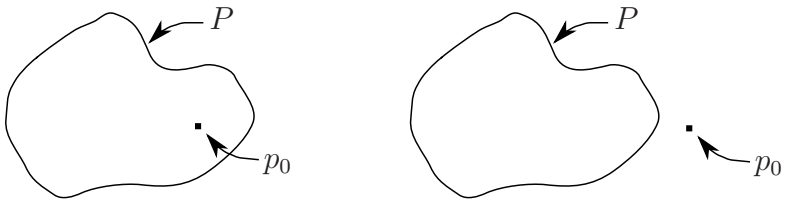
The classic loss for training CRFs is the conditional likelihood. Given a distribution p, and some particular training example $(\hat{\mathbf{y}}, \hat{\mathbf{x}})$, it is defined by

$$L(p, \hat{\mathbf{y}}, \hat{\mathbf{x}}) = -\log p(\hat{\mathbf{y}}|\hat{\mathbf{x}}).$$

If we are learning using this loss, we would seek the distribution p^* in P that has the best loss on the entire training set, i.e.,

$$p^* = \arg\min_{p \in P} \sum_{(\hat{\mathbf{y}}, \hat{\mathbf{x}})} L(p, \hat{\mathbf{y}}, \hat{\mathbf{x}}).$$

If the model is well-specified, p^* will converge to p_0 in the infinite data limit, using the



(a) A well-specified model.

(b) A misspecified model.

Figure 1.2: The true distribution p_0 may or may not be representable.

conditional likelihood, or other standard loss functions such as the pseudolikelihood. Of course, if $p_0 \notin P$, it is impossible to converge[2] to p_0. In practice, however, the true data-generating mechanism is usually unknown to some degree. If $p_0 \notin P$, we should not design a loss function to converge to p_0, but rather to the *best* distribution in P.

Purposive Loss Functions. The question, of course, is what is meant by "best"? There is no general answer. If forecasting the stock market, the best distribution might be the one that leads to the highest expected return. If predicting the scene in front of a robotic vehicle, we should like to avoid driving off cliffs.

It turns out that the conditional likelihood will asymptotically find the distribution that minimizes the expected Kullback-Leibler divergence from the true distribution (Section 3.2). In the infinite data limit, one will recover

$$p^* = \arg\min_{p \in P} \sum_{\mathbf{x}} p_0(\mathbf{x}) \mathrm{KL}\big(p_0(\mathbf{y}|\mathbf{x}) || p(\mathbf{y}|\mathbf{x})\big).$$

This is a reasonable criterion in many cases, but is not in general optimal.

Is it possible to define more purposive loss functions with out considering the details of the application? Consider how a graphical model will be used. Typically, one runs an inference algorithm on it. A key idea of this thesis is that if a model will only be used to produce marginals, then it only matters how accurate the marginals are. Even if p_0 is not in P, it is possible that some other distribution exists that still

[2]Strictly speaking, convergence is still possible if for every ϵ, there exists some $p \in P$ such that $d(p, p_0) < \epsilon$, for some appropriately defined distance measure d. A more correct definition would be to say the model is misspecified if there exists some ϵ such that for all $p \in P$, $d(p, p_0) > \epsilon$.

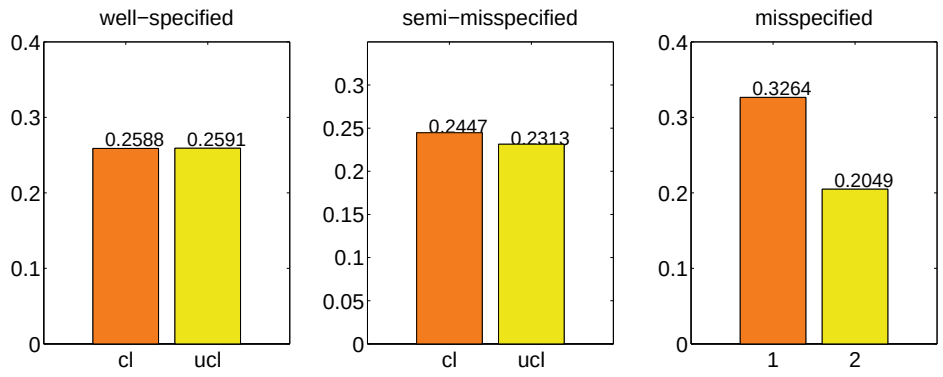


Figure 1.3: Test univariate classification error for three different datasets. cl: conditional likelihood training. ucl: univariate conditional likelihood training.

has marginals very close to those of p_0.

There are many ways to measure marginal accuracy. The simplest may be the *univariate* conditional likelihood, defined by

$$L(p, \hat{\mathbf{y}}, \hat{\mathbf{x}}) = -\sum_i \log p(\hat{y}_i|\hat{\mathbf{x}}),$$

which, assuming the global minimum is found, will converge in the infinite data limit to

$$p^* = \arg\min_{p \in P} \sum_{\mathbf{x}} p_0(\mathbf{x}) \sum_i \mathrm{KL}\big(p_0(y_i|\mathbf{x})||p(y_i|\mathbf{x})\big).$$

This loss only tries to measure how close the univariate marginals $p(y_i|\mathbf{x})$ are to the true marginals $p_0(y_i|\mathbf{x})$. In particular, it does not matter how close the joint distribution $p(\mathbf{y}|\mathbf{x})$ is to the true joint distribution $p_0(\mathbf{y}|\mathbf{x})$. There are other univariate loss functions, here called the "smoothed univariate classification error", and the "univariate quadratic loss". All these can be extended to the clique-wise case, measuring how close marginals like $p(\mathbf{y}_c|\mathbf{x})$ are to $p_0(\mathbf{y}_c|\mathbf{x})$.

Chapter 3 concludes with some experiments testing different loss functions on tractable (chain-like) models. Figure 1.3 shows a subset of the results. Here, the conditional likelihood and univariate conditional likelihood are both fit to three different

data sets. The *well-specified* dataset is generated (by Markov chain Monte Carlo) by a representable distribution with random parameters. In this case, the results of training on the two loss functions are nearly identical. The *semi-misspecified* and *misspecified* datasets are generated from distributions not obeying the conditional independencies asserted by the graph. (In the semi-misspecified case, these assumptions are only slightly violated.) In these cases, the univariate conditional likelihood is able to produce significantly more accurate univariate marginals than the conditional likelihood.

Though the univariate conditional likelihood was motivated here entirely by modeling concerns, we will see in later chapters that loss functions like this also enjoy certain *computational* advantages.

Chapter 4: Implicit Fitting

The second issue motivating new loss functions, intractable inference, is addressed in Chapters 4 and 5. The trouble, as discussed above, is that it is intractable to compute the marginals $p(y_i|\mathbf{x})$ or $p(\mathbf{y}_c|\mathbf{x})$ in general (high treewidth) graphs. For related reasons, it is not possible to fit the conditional likelihood.

Note that this intractability affects both the learning and inference stages. In learning, we cannot fit by the conditional likelihood or any of the univariate or clique-wise loss functions, as these require the marginals. As a thought experiment, however, suppose we *could*. This is not so unreasonable if large resources are available for the learning stage. For example, one could run an exhaustive Markov-chain Monte Carlo algorithm (e.g. Gibbs sampling) to closely approximate the marginals.

However, intractability arises again in the inference stage. This usually must be completed more quickly, forcing the use of approximations. Even if the marginals $p(y_i|\mathbf{x})$ or $p(\mathbf{y}_c|\mathbf{x})$ are very accurate, this accuracy is in some sense "wasted". The infer-

ence procedure will not return the true marginals, but rather approximate marginals $b(y_i|\mathbf{x})$ or $b(\mathbf{y}_c|\mathbf{x})$.

Fitting to Approximations. The strategy of the rest of the thesis is roughly this: Instead of fitting p, fit b. That is, rather than fitting the graphical model so that the marginals it produces under exact inference are accurate, fit so that the *approximate* marginals are. We can consider the parameters of a graphical model as simply defining a *mapping* from the input $\mathbf{x}$ to predicted marginals b. These marginals can be used in any of the univariate or clique-wise loss functions.

Chapter 4 considers the mapping defined by a convex inference procedure. Recall from above that Loopy Belief Propagation can be seen as optimizing a certain approximate energy function. This function happens to be non-convex in general, meaning it is difficult to reliably identify the global minimum. However, other similar approximations yield convex energy functions.

Performing Inference. For Chapter 4, it is convenient to phrase the LBP optimization (Eq. 1.3) more abstractly. If we notationally replace the functions $b(y_i|\mathbf{x})$ and $b(\mathbf{y}_c|\mathbf{x})$ with a single vector $\mathbf{b}$ containing all univariate and clique-wise beliefs, the LBP optimization can be rephrased and generalized to the form

$$\begin{aligned} \text{minimize} \quad & F(\mathbf{b}) && = \mathbf{w}(\mathbf{x})^T(\mathbf{b} \odot \log \mathbf{b}) + \mathbf{v}(\mathbf{x})^T\mathbf{b} \\ \text{s.t.} \quad & A\mathbf{b} && = \mathbf{d} \\ & \mathbf{b} && \geq \mathbf{0}. \end{aligned} \tag{1.6}$$

Appropriate choice of the matrix A and vector $\mathbf{d}$ will result in enforcing local consistency (Eqs. 1.4 and 1.5), while the vector $\mathbf{v}$ corresponds to the values $-\log \psi(\mathbf{y}_c, \mathbf{x})$. The vector $\mathbf{w}$ weights the entropy terms. Notice that if $\mathbf{w}$ is positive, this is a *convex* inference procedure. (If $\mathbf{w}$ is chosen to correspond to LBP inference, some entries of

w will be negative.)

Unfortunately, however, existing message passing algorithms cannot perform this general optimization. Chapter 4 suggests two (non message-passing) algorithms. The first is based on taking the Lagrange dual. This results in a relatively simple unconstrained maximization problem, where the gradient and Hessian are available in closed form. Experimentally, however, standard optimization algorithms, including L-BFGS, Newton's method, and nonlinear Conjugate Gradients take more iterations and computation time than the primal solution. The primal algorithm is based on a novel successive approximation scheme. The terms $b_i \log b_i$ are replaced with a quadratic upper bound. This bound is simultaneously used to asymptotically enforce the constraint $b_i \geq 0$. As such, each iteration corresponds to solving a quadratic optimization under linear constraints, and so can be reduced to a single linear system. In practice, this algorithm typically converges to high accuracy in 5-20 iterations.

Fitting Mappings. Since convex functions can be minimized quite reliably, we can think of a model and a convex inference procedure together producing a *mapping* from an input to predicted marginals. So, acting on the philosophy above, we would like to fit this mapping to be accurate. Put another way, we would like to shape the energy function (over predicted marginals) so as to put the minima in good places.

Clearly, given the parameters defining a model, it is not hard to *measure* a marginal-based loss function: One simply performs inference to get predicted marginals, then plugs these into a loss function. In principle, this is enough to allow learning, but things would be much easier if one could also compute the *gradient* of the loss with respect to the parameters of the model. This appears difficult to do, since the model determines the loss only though an optimization.

Now, take some parameter of the model θ. We want to know how changing θ will affect the loss. However, the loss is determined by the intermediate step of computing beliefs. By the vector chain rule, the derivative of the loss with respect to θ can be

decomposed as

$$\frac{dL}{d\theta} = \frac{dL}{d\mathbf{b}}^T \frac{d\mathbf{b}}{d\theta}. \tag{1.7}$$

The first term, the derivative of the loss with respect to beliefs is trivial to calculate. However, the value of $\mathbf{b}$ is determined by solving the optimization problem in Eq. 1.6. Thus, the second term cannot be calculated directly. The following self-contained result enables learning:

Claim: If $\mathbf{b}(\theta) = \arg\min_{\mathbf{b}} F(\mathbf{b}, \theta)$ such that $A\mathbf{b} = \mathbf{d}$, then

$$\frac{d\mathbf{b}}{d\theta} = \left(D^{-1}A^T(AD^{-1}A^T)^{-1}AD^{-1} - D^{-1}\right)\frac{\partial^2 F}{\partial \mathbf{b}\partial\theta},$$

where $D := \frac{\partial^2 F}{\partial \mathbf{b}\partial \mathbf{b}^T}$.

This tells us how the predicted beliefs will change if we modify the parameter θ, since $\partial^2 F/\partial\mathbf{b}\partial\theta$ will be known in closed form. These derivatives can be used to compute $dL/d\theta$, i.e. the derivatives of the loss function. This, in turn, can be fed into a nonlinear optimization to improve the loss on data.

Chapter 5: Procedural Fitting

An advantage of LBP inference is that running it for a few iterations is very fast, and scalable to huge problems. Often, just a few iterations give acceptable accuracy. It is also sometimes suggested that if the nonconvex LBP approximation is minimized correctly, it gives better predictions than convex procedures. Unfortunately, Chapter 4's approach cannot handle non-convex inference.

Procedures as Mappings. This chapter pursues a different strategy. Rather than fitting an inference *energy function*, fit a message passing *inference procedure*.

In order to think of a message-passing procedure as a mapping, several things must be held constant: The beliefs must be initialized in the same way, updates must take place in the same order, and for the same number of iterations.

Again, the technical problem arises to calculate the gradient of a loss with respect to the parameters of the model. Clearly, the loss is differentiable: it is calculated from the final beliefs, which are the product of a fixed series of basic differentiable operations (addition, multiplication). It is possible to analytically derive an algorithm to "backpropagate" the derivatives of a loss function to find how it changes with respect to the parameters of the model. Here, however, the far simpler (and equivalent) solution is taken of using automatic differentiation tools to do this automatically.

Again, with the derivatives $dL/d\theta$ in hand, one can apply a nonlinear optimization algorithm to fit the parameters of the model to improve the loss.

Chapter 6: Experiments

Chapter 6 finally applies the above strategies to data.

Binary Digits. The first dataset consists of binary images of handwritten digits that have been corrupted by various amounts of noise. The goal is to recover the original image. The advantages of this dataset are simplicity, and the ability to smoothly vary the amount of noise, ranging from an easy problem with low noise, to a rather difficult one with high noise. Some example results are presented in Figure 1.4 and Table 1.1. This compares a convex approximation of the conditional likelihood (`convex`), implicit fitting of the univariate conditional likelihood (`ucl`), and procedural fitting of same, with 4 iterations of updates (`ucl-4`). We see, roughly, that at low amounts of noise, all three methods perform well, while with large amounts of noise, the proposed methods have an edge, with procedural fitting performing better than implicit fitting.

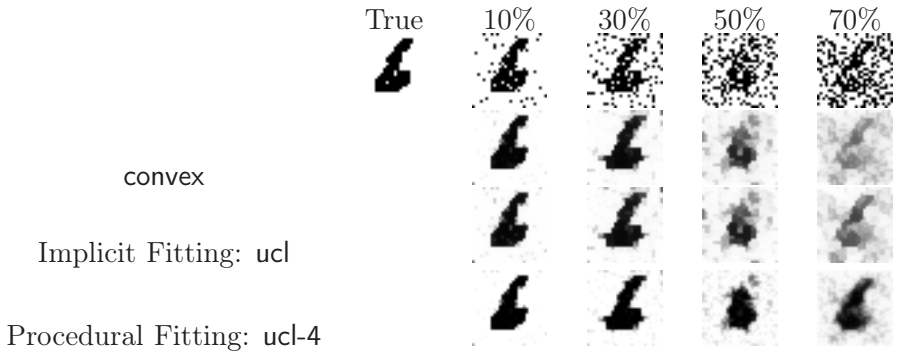


Figure 1.4: Some example binary digit results.

	10%	30%	50%	70%
`convex`	.0092	.0304	.0716	.158
Implicit Fitting: `ucl`	.0104	.0319	.0710	.135
Procedural Fitting: `ucl-4`	.0087	.0285	.0599	.105

Table 1.1: Example binary digit univariate classification errors.

Fitting Entropy Parameters. An unusual idea explored here is to adjust entropy parameters when doing implicit fitting (Section 6.3) for best performance. This appears to significantly improve the performance of the implicit fitting strategy. For example, it achieves a univariate classification error of around .059 on the binary digits with 50% noise, very similar to the error of procedural fitting.

StreetScenes. The next dataset, known as StreetScenes, consists of unstructured images taken on streets around Massachusetts. The goal is to label each pixel. Here, five classes are used: buildings, trees, roads/sidewalks, cars, and sky. Due to the presence of unlabeled pixels, the convex likelihood cannot be used. Instead, it is necessary to compare to a poorer approximation of the likelihood, the pseudolikelihood. The proposed methods deal easily with unlabeled data, with no extra technique required-one simply computes the loss functions over the observed variables.

Figure 1.5 shows some example results on the StreetScenes dataset. The baseline result corresponds to fitting a totally disconnected graphical model, where each pixel independently predicts its label given the surrounding image patch. The `uquad`

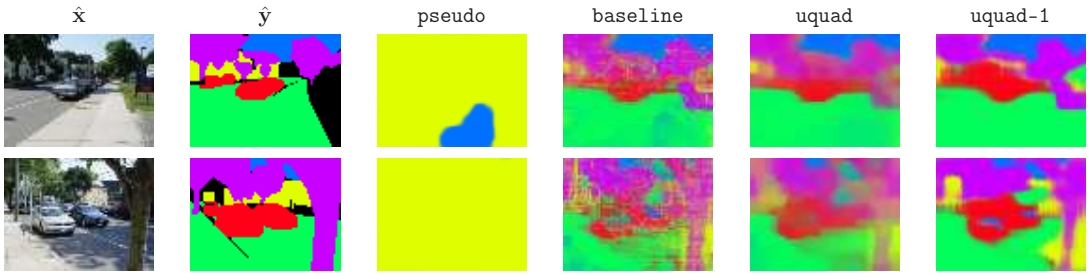


Figure 1.5: Some example results on the StreetScenes dataset.

(implicit fitting) and `uquad-1` (procedural fitting) results improve on this, while the pseudolikelihood actually performs far worse.

Chapter 7: Discussion

Chapter 7 discusses previous work, future work, and considers the strengths and weaknesses of the proposed methods.

Summary

Thesis: Implicit and procedural fitting can adjust parameters of graphical models to optimize the performance of approximate marginal inference methods.

Many problems have a certain character, where one cares mostly about the average results per variable, or on small groups of variables. Problems like these are most naturally addressed with marginal inference. In general graphs, however, approximate inference must be used. Implicit and procedural fitting are two methods for fitting graphical models that try to compensate for the defects in approximate inference. These methods directly seek the parameters that will yield the most accurate *predictions.* These methods are only guaranteed to find a local minima in parameter space.

Contributions. The two major contributions of this thesis, clearly, are the implicit fitting and procedural fitting methods. Other specific technical contributions are:

1. The univariate quadratic loss (Section 3.5.2).

2. The clique-wise extension of the univariate loss functions, i.e. the clique-wise conditional likelihood, quadratic loss, and smoothed classification error, along with an argument for their consistency. (Section 3.6).

3. The algorithms for optimizing beliefs, either by the dual (Section 4.3), or by a successive approximation scheme in the primal (Section 4.4).

Chapter 2

Background on Graphical Models

This thesis will make heavy use of somewhat non-standard notation. As is common, boldface denotes a vector. We also allow vectors with "set subscripts". For example, if $\mathbf{x} = (x_1, ..., x_5)$, and $c = \{2, 4, 5\}$, then $\mathbf{x}_c = (x_2, x_4, x_5)$. Similarly, conditions can act as subscripts. For example, $\mathbf{x}_{\geq 4} = (x_4, x_5)$, and $\mathbf{x}_{\neq 2} = (x_1, x_3, x_4, x_5)$.

2.1 Graphical Models

At its most basic, a graphical model is simply a way of writing a probability distribution in a *factorized* form. Consider some probability distribution,

$$p(\mathbf{x}) = p(x_1, x_2, \cdots, x_n).$$

If each variable x_i is binary, there are 2^n possible configurations for $\mathbf{x}$. Clearly, the "brute-force" approach of writing down the probability of each configuration will fail for any reasonable n. Instead, in a graphical model, one writes the joint distribution as a product of terms, each defined over some (presumably small) subset of variables. This will resist the "curse of dimensionality" as long as each subset is of a bounded size.

Clearly, not every joint distribution can be written as a product of terms. When will it be possible? Graphical models give theoretical guarantees in terms of *conditional independencies*. The exact nature of the conditional independence assumptions leads to different types of graphical models.

2.1.1 Directed Models

By elementary rules of probability, any distribution can be written exactly as a product of terms, where each term is the probability of one variable, given all those before it in some order.

$$p(\mathbf{x}) = p(x_1)p(x_2|x_1)\cdots p(x_n|x_1, x_2, \cdots, x_{n-1})$$

In a directed model, or Bayesian Network, one assumes a set of "parents" for each variable that render it independent of all others before it in the ordering. Let $\pi(i)$ denote the set of parents for x_i. Then, the assumption is that

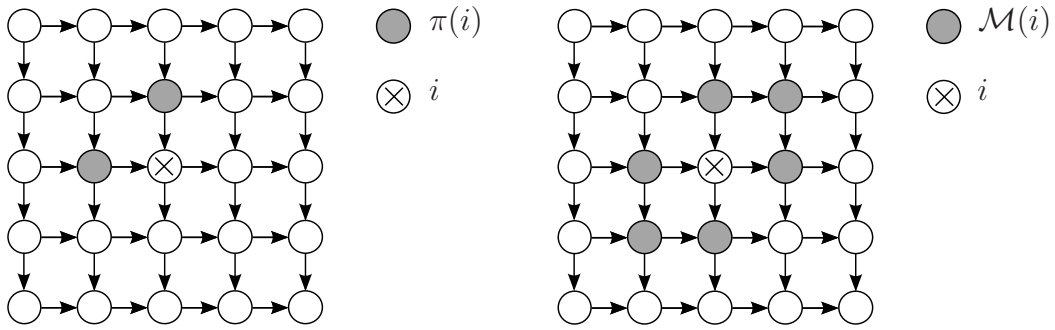
$$p(x_i|x_1, \cdots, x_{i-1}) = p(x_i|\mathbf{x}_{\pi(i)}) \tag{2.1}$$

and so

$$p(\mathbf{x}) = \prod_i p(x_i|\mathbf{x}_{\pi(i)}).$$

Given a set of assumed parents for each node, it is convenient to picture the situation by drawing a graph with one node for each variable, and directed edges from each parent to each child (Fig. 2.1(a)).

Notice that there is no reference in Eq. 2.1 to any conditional independence relative to variables x_{i+1}, x_{i+2}, $\cdots$, x_n. A directed model asserts only independence of a variable to those *before* it, given its parents. What, then, is the "Markov blanket" of x_i– the set of variables that render it independent of *all* others? This turns out to



(a) A variable and its parents. (b) A variable and its Markov blanket $\mathcal{M}$.

Figure 2.1: Directed models.

consist of x_i's parents, children, and the parents of its children (Fig. 2.1(b)).

Though the ideas of this thesis apply to all graphical models, the presentation is confined to one type for clarity. Hence, this thesis will not focus on directed models.

2.1.2 Markov Random Fields

In a Markov Random Field (MRF), one directly specifies the Markov blanket of each variable. Let $\mathcal{N}(i)$ denote the set of "neighbors" of i. One then asserts that x_i is independent of all other variables given its neighbors.

$$p(x_i|\mathbf{x}_{\neq i}) = p(x_i|\mathbf{x}_{\mathcal{N}(i)})$$

The neighborhood system must be symmetric.

$$i \in \mathcal{N}(j) \leftrightarrow j \in \mathcal{N}(i)$$

An MRF is pictured by drawing a graph with one node for each variable, and undirected edges between all neighbors (Fig. 2.2).

The immediate question is, given an MRF, what form can its probability distribution take? It is not easy to specify $p(\mathbf{x})$ in terms of local conditional distributions, because in general graphs these turn out to have severe, non-obvious constraints. The

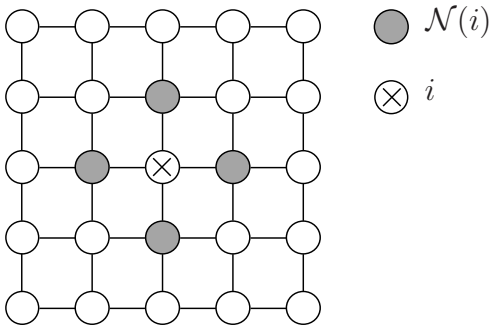


Figure 2.2: An undirected model: A variable and its neighbors.

solution ultimately came in the form of the Hammersley-Clifford theorem. Notice that it is required that p give positive probability to all configurations.

> **Hammersley-Clifford Theorem:** $p(\mathbf{x}) > 0$ obeys the set of conditional independencies asserted by a graph if and only if there exist functions $\psi(\mathbf{x}_c)$ such that
>
> $$p(\mathbf{x}) = \frac{1}{Z} \prod_c \psi(\mathbf{x}_c), \tag{2.2}$$
>
> where the product is over the set of *cliques* c in the neighborhood graph, and Z is a normalization constant.

$$Z = \sum_{\mathbf{x}} \prod_c \psi(\mathbf{x}_c). \tag{2.3}$$

The local functions $\psi(\mathbf{x}_c)$ do not have a direct probabilistic interpretation.

Note that while it is easy to show that a distribution written as in Eq. 2.2 obeys the conditional independencies asserted by a graph, the converse is not at all obvious. The following is based upon the original proof by Besag [3]. Here, it is assumed that each variable x_i can take on a finite number of values, $0, \cdots, M$. The vector of all zeros is denoted by $\mathbf{0}$.

Proof Sketch:

1. Define $q(\mathbf{x}) = \log \frac{p(\mathbf{x})}{p(\mathbf{0})}$

2. There is a *unique* expansion for $q(\mathbf{x})$ of the form

$$q(\mathbf{x}) = \sum_i x_i g_i(x_i) + \sum_i \sum_{j \neq i} x_i x_j g_{ij}(x_i, x_j) + \cdots + x_1 \cdots x_N g_{1\cdots N}(x_1, \cdots, x_N). \tag{2.4}$$

3. Define $\mathbf{x}_{i\to 0} = (x_1, \cdots, x_{i-1}, 0, x_{i+1}, \cdots, x_N)$. Then

$$q(\mathbf{x}) - q(\mathbf{x}_{i\to 0}) = \log p(x_i|\mathbf{x}_{\mathcal{N}(i)}) - \log p(x_i = 0|\mathbf{x}_{\mathcal{N}(i)}).$$

   In particular, note this is a function only of x_i, and $\mathbf{x}_{\mathcal{N}(i)}$.

4. At the same time, (with x_1 chosen arbitrarily for convenience)

$$q(\mathbf{x}) - q(\mathbf{x}_{1\to 0}) = x_1\big(g_1(x_1) + \sum_{j\neq 1} x_j g_{1j}(x_1, x_j) + \cdots + x_2 \cdots x_N g_{1\cdots N}(x_1, \cdots, x_N)\big).$$

5. So, $g_c \neq 0$ only if $\forall i, j \in c$, $i \in N(j)$. That is, all nonzero functions in the expansion of Eq. 2.4 are functions of *cliques*.

Additional discussion for each of the above steps follows.

Step 1: Note that $q(\mathbf{x})$ is just a function, not a valid probability distribution.

Step 2: Suppose that an expansion exists. To see that it is unique, note for example that $q(0, \cdots, 0, x_i, 0, \cdots, 0) = x_i g_i(x_i)$, since all other terms on the right hand side of Eq. 2.4 will be zero. This fixes all terms of the form $x_i g_i(x_i)$. Next, one can consider values like $q(0, \cdots, 0, x_i, 0, \cdots, 0, x_j, 0, \cdots, 0)$, which will then fix all the functions $x_i x_j g_{ij}(x_i, x_j)$. Analogous values then fix all functions up to on the right-hand side of Eq. 2.4. Since every value $q(\mathbf{x})$ will be satisfied by this strategy, clearly an expansion *does* exist.

Step 3: This is quite easy to show.

$$\begin{aligned} q(\mathbf{x}) - q(\mathbf{x}_{i\to 0}) &= \log\frac{p(\mathbf{x})}{p(\mathbf{0})} - \log\frac{p(\mathbf{x}_{i\to 0})}{p(\mathbf{0})} \\ &= \log\frac{p(\mathbf{x})}{p(\mathbf{x}_{i\to 0})} \\ &= \log\frac{p(\mathbf{x}_{\neq i})p(x_i|\mathbf{x}_{\neq i})}{p(\mathbf{x}_{\neq i})p(x_i=0|\mathbf{x}_{\neq i})} \\ &= \log\frac{p(x_i|\mathbf{x}_{\mathcal{N}(i)})}{p(x_i=0|\mathbf{x}_{\mathcal{N}(i)})} \end{aligned}$$

Step 4: Simply notice that in the expansions for $q(\mathbf{x})$ and $q(\mathbf{x}_{1\to 0})$, any terms not involving x_1 will cancel.

Step 5: In the expression for step 4, consider g_{1j}, $j \notin \mathcal{N}(1)$. Take the value $\mathbf{x}^* = (x_1, 0, \cdots, 0, x_j, 0, \cdots, 0)$. Observe that

$$q(\mathbf{x}^*) - q(\mathbf{x}^*_{1\to 0}) = x_1\big(g_1(x_1) - x_j g_{1j}(x_1, x_j)\big).$$

But, by step 3, we know that $q(\mathbf{x}^*) - q(\mathbf{x}^*_{1\to 0})$ is independent of x_j. So we must have $x_1 x_j g_{1j} = 0$. Similarly, consider g_{1jk}, $j \notin \mathcal{N}(1)$. Take the value

$$\mathbf{x}^* = (x_1, 0, \cdots, 0, x_j, 0, \cdots, 0, x_k, 0, \cdots, 0).$$

Now,

$$q(\mathbf{x}^*) - q(\mathbf{x}^*_{1\to 0}) = x_1\big(g_1(x_1) + x_j g_{1j}(x_1, x_j) + x_k g_{1k}(x_1, x_k) + x_j x_k g_{1jk}(x_1, x_j, x_k)\big).$$

Again, by step 3, we know that $q(\mathbf{x}^*) - q(\mathbf{x}^*_{1\to 0})$ is independent of x_j. Since $x_1 x_j g_{1j}$ is already know to be zero, this implies that $x_1 x_j x_k g_{1jk} = 0$. Similar examples hold for higher-order functions.□

2.1.3 Conditional Random Fields

Suppose we want to represent a conditional distribution $p(\mathbf{y}|\mathbf{x})$. A Conditional Random Field (CRF) is defined by

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_c \psi(\mathbf{y}_c, \mathbf{x}), \tag{2.5}$$

with the normalization constant now a function of $\mathbf{x}$.

$$Z(\mathbf{x}) = \sum_{\mathbf{y}} \prod_c \psi(\mathbf{y}_c, \mathbf{x})$$

One way to arrive at this definition is to take a Markov Random Field defined jointly over $\mathbf{y}$ and $\mathbf{x}$, and then condition it.

$$p(\mathbf{y}, \mathbf{x}) = \frac{1}{Z} \prod_c \psi(\mathbf{y}_c, \mathbf{x}_c) \rightarrow p(\mathbf{y}|\mathbf{x}) = \frac{p(\mathbf{y}, \mathbf{x})}{p(\mathbf{x})} = \frac{\prod_c \psi(\mathbf{y}_c, \mathbf{x}_c)}{\sum_{\mathbf{y}'} \prod_c \psi(\mathbf{y}'_c, \mathbf{x}_c)} \tag{2.6}$$

Note that if there are any cliques c that contain only variables in $\mathbf{x}$ they can be dropped, since those terms will be constant on the numerator and denominator in Eq. 2.6.

The definition of a CRF in Eq. 2.5 is slightly more general since it allows each term to depend on the entire vector $\mathbf{x}$ rather than just the variable in a clique.

2.2 Inference in Graphical Models

Suppose we have a graphical model. What will we do with it? From a decision theory viewpoint, there is no universal answer. An application demands a decision, and the graphical model will be used in a problem-dependent way to make the choice with the best expected outcome, taking into account how risk adverse we are, etc. Nevertheless, there are several common questions to ask of graphical models, called "inference" problems.

In maximum a posteriori probability or MAP inference, one looks for the single value $\mathbf{y}$ with maximum probability given $\mathbf{x}$.

$$\textbf{MAP inference:} \quad \mathbf{y}^* = \arg\max_{\mathbf{y}} p(\mathbf{y}|\mathbf{x}) \tag{2.7}$$

Another common problem, and the one of focus in this thesis is marginalization.

$$\textbf{Marginal inference:} \quad p(y_i^*|\mathbf{x}) = \sum_{\mathbf{y}:y_i=y_i^*} p(\mathbf{y}|\mathbf{x}) \tag{2.8}$$

While MAP inference looks like (and is) a challenging combinatorial optimization problem, the notational simplicity of writing down $p(y_i|\mathbf{x})$ understates the difficulty of computing marginals. The naive method– a brute force sum over all vectors $\mathbf{y}$: $y_i = y_i^*$ as in Eq. 2.8– will rarely be tractable, due to the curse of dimensionality.

A problem that has aspects of both of the above is Maximum Posterior Marginal or MPM inference[4].

$$\textbf{MPM inference:} \quad y_i^* = \arg\max_{y_i} p(y_i|\mathbf{x}) \tag{2.9}$$

Whereas MAP inference pursues the joint vector $\mathbf{y}^*$ that has maximum probability, MPM inference does this separately for each variable. Thus, if one cares about the *number of variables in error*, as opposed to if *all variables are correct simultaneously* or not, MPM is to be preferred to MAP inference. In MPM inference, one first runs a marginalization algorithm, then for each index i chooses the value with maximum marginal probability. Since the second step is trivial, the main computational problem remains marginal inference.

The focus of this thesis is marginal inference, both for its own sake, and for enabling MPM inference. The discussion of inference algorithms below will be entirely

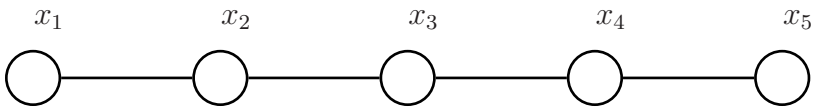


Figure 2.3: A Chain MRF

for marginal inference. However, the computational issues faced by MAP inference are similar, and sometimes almost the same algorithm can be used for both with slight changes. (An example of this is how the "sum-product" version of belief propagation discussed below for marginal inference can be trivially transformed into a "min-sum" form for MAP inference. [5, 26.2-3])

2.2.1 Dynamic Programming

This section will first show how marginals can be computed exactly on pairwise chains by dynamic programming. The next section generalizes this to arbitrary singly-connected graphs through a message-passing algorithm. For simplicity of notation, this section computes the marginals $p(x_i)$ for an MRF, but with trivial changes the same algorithm can compute $p(y_i|\mathbf{x})$ for a CRF.

Consider a chain MRF defined on pairs of variables (Fig. 2.3).

$$p(\mathbf{x}) \quad \propto \quad \prod_{i=1}^{n-1} \psi(x_i, x_{i+1})$$

Since, by definition $p(x_i) = \sum_{\mathbf{x}_{\neq i}} p(\mathbf{x})$,

$$\begin{aligned} p(x_i) \quad &\propto \quad \sum_{\mathbf{x}_{<i}} \sum_{\mathbf{x}_{>i}} \prod_{j=1}^{i-1} \psi(x_j, x_{j+1}) \prod_{j=i}^{n-1} \psi(x_j, x_{j+1}) \\ &= \quad \Big(\sum_{\mathbf{x}_{<i}} \prod_{j=1}^{i-1} \psi(x_j, x_{j+1})\Big)\Big(\sum_{\mathbf{x}_{>i}} \prod_{j=i}^{n-1} \psi(x_j, x_{j+1})\Big) \qquad (2.10) \\ &= \quad T_L(x_i) T_R(x_i) \qquad (2.11) \end{aligned}$$

where the tables T_L and T_R are defined for the sums two sums above. These tables can be computed efficiently by dynamic programming.

$$T_L(x_i) := \sum_{\mathbf{x}_{<i}} \prod_{j=1}^{i-1} \psi(x_j, x_{j+1}) \qquad T_R(x_i) := \sum_{\mathbf{x}_{>i}} \prod_{j=i}^{n-1} \psi(x_j, x_{j+1})$$

$$= \sum_{x_{i-1}} \psi(x_{i-1}, x_i) T_L(x_{i-1}) \quad (2.12) \qquad = \sum_{x_{i+1}} \psi(x_i, x_{i+1}) T_R(x_{i+1}) \quad (2.13)$$

Boundary conditions simply use $T_L(x_1) = T_R(x_n) = 1$.

We can also use these tables to compute pairwise probabilities as

$$\begin{aligned} p(x_i, x_{i+1}) &\propto \big(\sum_{\mathbf{x}_{<i}} \prod_{j=1}^{i-1} \psi(x_j, x_{j+1})\big) \psi(x_i, x_{i+1}) \big(\sum_{\mathbf{x}_{>i+1}} \prod_{j=i+1}^{n-1} \psi(x_j, x_{j+1})\big) \\ &= T_L(x_i) \psi(x_i, x_{i+1}) T_R(x_{i+1}). \end{aligned}$$

If computed exactly as in Eqs. 2.12 and 2.13, the values in the tables will often become very large or small. However, we can observe that the values are only actually needed up to a constant factor. Thus, to avoid numerical problem, one can instead use updates like $T_L(x_i) = \alpha \sum_{x_{i-1}} T_L(x_{i-1}) \psi(x_{i-1}, x_i)$, where α is chosen, e.g., so that $\sum_{x_i} T_L(x_i) = 1$.

2.2.2 Belief Propagation

The Belief Propagation or Sum-Product algorithm can handle general tree graphs, and is more elegant than dynamic programming. Instead of "tables", this algorithm sends "messages" performing the same function. There are two types of messages, those from variables to cliques,

$$m_{i \to c}(x_i) = \prod_{d : i \in d, d \neq c} m_{d \to i}(x_i), \tag{2.14}$$

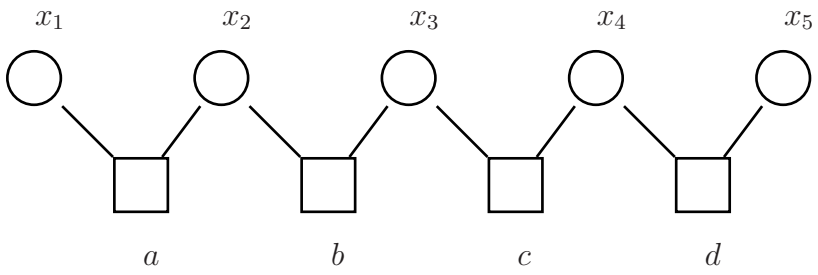


Figure 2.4: A Chain Factor Graph

and those from cliques to variables.

$$m_{c\to i}(x_i) = \sum_{\mathbf{x}_{c\setminus i}} \psi(\mathbf{x}_c) \prod_{j\in c, j\neq i} m_{j\to c}(x_j) \tag{2.15}$$

The final univariate marginals can be recovered from

$$p(x_i) \propto \prod_{d: i\in d, d\neq c} m_{d\to i}(x_i),$$

while the clique marginals are given by

$$p(\mathbf{x}_c) \propto \psi(\mathbf{x}_c) \prod_{j\in c} m_{j\to c}(x_j).$$

This algorithm is best explained by example. It is convenient to picture a "factor graph" consisting again of nodes for variables, but now also square nodes for cliques. A node and clique are connected if and only if the node participates in the clique. First, let us consider again a chain. The factor graph in Fig. 2.4 corresponds to Fig. 2.3 above.

Messages are not "sent" until all prerequisite messages have been "received". Note that the message $m_{1\to a}(x_1) = 1$ can immediately be sent. The messages can then be computed from left to right in order: $m_{a\to 2}, m_{2\to b}, m_{b\to 3}$, etc. We then find that $m_{a\to 2}(x_2) = \sum_{x_1} \psi(x_1, x_2)$, and more generally that the messages $m_{c\to i}(x_i)$ coming from the left correspond exactly to the values $T_L(x_i)$ in dynamic programming, while

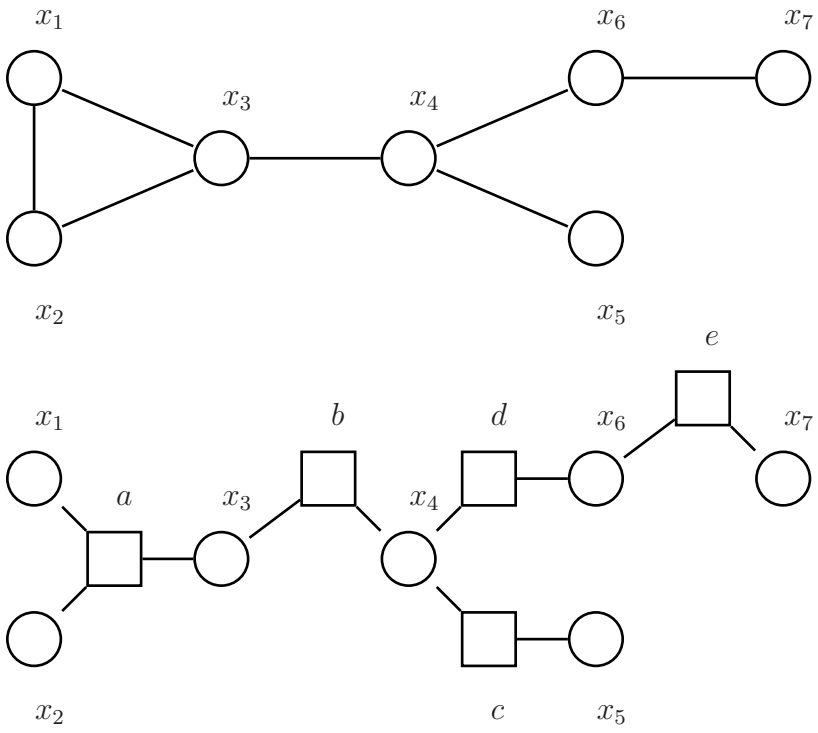


Figure 2.5: A general graph, at top pictured as an MRF, and at bottom as a Factor Graph.

those coming from the right correspond to $T_R(x_i)$. This justifies the algorithm at least for the case of pairwise chain graphs.

Now consider the more general graph in Fig. 2.5. Again, the messages from nodes at the end will be constant, so $m_{1\to a}(x_1) = m_{2\to a}(x_1) = 1$. It follows that $m_{a\to 3}(x_3) = \sum_{x_1}\sum_{x_2}\psi(x_1, x_2, x_3)$. Similarly, it is not hard to see that each message from one element to another consists of a sum over all configurations "on the other side" of the sender. For example

$$m_{4\to d}(x_4) = \sum_{\mathbf{x}_{\{1,2,3,5\}}} \psi(\mathbf{x}_{\{1,2,3\}})\psi(\mathbf{x}_{\{3,4\}})\psi(\mathbf{x}_{\{4,5\}}),$$

and

$$m_{b\to 3}(x_3) = \sum_{\mathbf{x}_{\{4,5,6,7\}}} \psi(\mathbf{x}_{\{4,5\}})\psi(\mathbf{x}_{\{4,6\}})\psi(\mathbf{x}_{\{6,7\}}).$$

As with dynamic programming, messages are only needed up to a constant factor. Thus, to prevent numerical problems, the update formulas (Eqs. 2.14 and 2.15) can introduce a normalizer chosen so that messages sum to one.

Notice that the belief propagation algorithm will be applicable exactly when the factor graph is singly connected. As in Fig. 2.5, the MRF need not be singly connected.

2.2.3 Computational Complexity and the Junction Tree Algorithm

What to do if faced with a graph that is not a tree? One possibility, if the graph is *nearly a tree*, is to convert it into an equivalent, singly connected graph. This can be done, roughly speaking, by creating variables with one state for each joint configuration of a set of variables in the original graph. Inference can proceed exactly on this new graph by Belief Propagation. The problem with this is that in a large general graph, a large number of variables will need to be aggregated, with an exponential number of joint configurations. Thus, the junction tree algorithm is only practical on graphs with low treewidth.

2.2.4 Loopy Belief Propagation and the Bethe Approximation

Though the belief propagation algorithm is defined for singly-connected graphs, with slight changes, one can *run* the algorithm on an arbitrary graph. The major difference is that the messages need to be initialized somehow, and iteratively re-updated until everything (hopefully) converges. This "loopy" belief propagation often appears to give good results, though it might converge to different stationary conditions, and might fail to converge at all.

A theoretical understanding of this algorithm was given by Yedidia et al. [6], who

made connections to an approximation from statistical physics known as the Bethe approximation. A rough idea of this is the following. Consider minimizing the KL-divergence between some "belief" distribution $b(\mathbf{x})$ and the true distribution $p(\mathbf{x}) = \frac{1}{Z}\prod_c \psi(\mathbf{x}_c)$. If done exactly, of course, we would simply recover $b = p$. However, it is useful to cast the problem in this "variational" form, since approximations can then be made to the optimization. It can easily be shown[1] that

$$\arg\min_b KL\big(b(\mathbf{x})||p(\mathbf{x})\big) = \arg\min_b \sum_{\mathbf{x}} b(\mathbf{x}) \log b(\mathbf{x}) - \sum_c \sum_{\mathbf{x}_c} b(\mathbf{x}_c) \log \psi(\mathbf{x}_c). \quad (2.16)$$

The second term is easy to compute exactly. The first term, known as the entropy, is difficult. In general, the curse of dimensionality prevents even representing an arbitrary distribution $b(\mathbf{x})$. To get a tractable approach, one can approximate the entropy with a function of local beliefs. The Bethe approximation is

$$\sum_{\mathbf{x}} b(\mathbf{x}) \log b(\mathbf{x}) \approx \sum_c \sum_{\mathbf{x}_c} b(\mathbf{x}_c) \log b(\mathbf{x}_c) + \sum_i n_i \sum_{x_i} b(x_i) \log b(x_i), \quad (2.17)$$

with $n_i = 1 - |\{c : i \in c\}|$. The motivation for this choice is that for singly-connected graphs, the Bethe approximation is exact[2]. So, under this approximation, one searches

[1] By definition, $\arg\min_b KL\big(b(\mathbf{x})||p(\mathbf{x})\big) = \arg\min_b \sum_{\mathbf{x}} b(\mathbf{x}) \log b(\mathbf{x}) - \sum_{\mathbf{x}} b(\mathbf{x}) \log p(\mathbf{x})$. Now, working on the right hand side, we can see that

$$\sum_{\mathbf{x}} b(\mathbf{x}) \log p(\mathbf{x}) = \sum_{\mathbf{x}} \sum_c b(\mathbf{x}) \log \psi(\mathbf{x}_c) - \sum_{\mathbf{x}} b(\mathbf{x}) \log Z$$

The second term can be disregarded, since it is a constant. Finally, make the substitution

$$\sum_{\mathbf{x}} \sum_c b(\mathbf{x}) \log \psi(\mathbf{x}_c) = \sum_c \sum_{\mathbf{x}_c} b(\mathbf{x}_c) \log \psi(\mathbf{x}_c).$$

[2] To see this, first note that a tree structured distribution can be written as

$$p(\mathbf{x}) = \prod_i p(x_i) \prod_c \frac{p(\mathbf{x}_c)}{\prod_{i \in c} p(x_i)}.$$

This is easy to see by induction, starting from a single clique and adding neighboring cliques one at a time. From this it follows that

for beliefs to minimize

$$\sum_c \sum_{\mathbf{x}_c} b(\mathbf{x}_c) \log b(\mathbf{x}_c) + \sum_i n_i \sum_{x_i} b(x_i) \log b(x_i) - \sum_c \sum_{\mathbf{x}_c} b(\mathbf{x}_c) \log \psi(\mathbf{x}_c).$$

However, for arbitrary local distributions $b(\mathbf{x}_c)$ and $b(x_i)$, there need not exist a consistent joint distribution $b(\mathbf{x})$ giving them. "Local consistency" means enforcing only that local beliefs are valid (i.e. non-negative and sum to one), and that clique beliefs marginalize to univariate beliefs.

$$\begin{aligned} b(x_i) \geq 0 \qquad & b(\mathbf{x}_c) \geq 0 \\ \sum_{x_i} b(x_i) = 1 \qquad & \sum_{\mathbf{x}_c} b(\mathbf{x}_c) = 1 \\ b(x_i) = \sum_{\mathbf{x_{c \setminus i}}} & b(\mathbf{x}_c) \end{aligned} \tag{2.18}$$

These constraints are sufficient for global consistency in singly-connected graphs, but not in general graphs. Moreover, the set of constraints needed to ensure global consistency in general graphs is intractably large [7].

Yedidia et al. [6] showed that if loopy belief propagation converges, it is at a stationary point of the Bethe free energy subject to local consistency and that a stationary point of the Bethe free energy subject to local consistency corresponds to a convergent point of loopy belief propagation[3].

It is important to emphasize there are two different approximations made by loopy

$$\begin{aligned} \sum_{\mathbf{x}} p(\mathbf{x}) \log p(\mathbf{x}) &= \sum_{\mathbf{x}} p(\mathbf{x}) \sum_c \log p(\mathbf{x}_c) + \sum_{\mathbf{x}} p(\mathbf{x}) \sum_i n_i \log p(x_i) \\ &= \sum_c \sum_{\mathbf{x}} p(\mathbf{x}_c) \log p(\mathbf{x}_c) + \sum_i n_i \sum_{\mathbf{x}} p(x_i) \log p(x_i) \end{aligned}$$

[3]To see this, form a Lagrangian enforcing constraints $b(x_i) = \sum_{\mathbf{x}_{c \setminus i}} b(\mathbf{x}_c)$, $\sum_{x_i} b(x_i) = 1$, and $\sum_{\mathbf{x}_c} b(\mathbf{x}_c) = 1$.

belief propagation:

1. The approximation of the entropy (Eq. 2.17).

2. The relaxation of global consistency into local consistency (Eq. 2.18).

Recall still that LBP may not converge, and that it can converge to different stationary points. Empirically, it is found that fixed points almost always correspond to local minima.

2.2.5 Mean Field

(This section can be skipped with out loss of continuity.) Mean field is an alternative method for approximate inference. The idea, as above, takes the perspective of

$$\mathcal{L} = \sum_c \sum_{\mathbf{x}_c} b(\mathbf{x}_c) \log b(\mathbf{x}_c) + \sum_i n_i \sum_{x_i} b(x_i) \log b(x_i) - \sum_c \sum_{\mathbf{x}_c} b(\mathbf{x}_c) \log \psi(\mathbf{x}_c)$$
$$+ \sum_c \sum_{i \in c} \sum_{x_i} \lambda_c(x_i) \big(b(x_i) - \sum_{\mathbf{x}_{c \setminus i}} b(\mathbf{x}_c) \big) + \sum_i \gamma_i (\sum_{x_i} b(x_i) - 1) + \sum_c \gamma_c (\sum_{\mathbf{x}_c} b(\mathbf{x}_c) - 1)$$

First, we show that a fixed point of this Lagrangian corresponds to a convergent configuration for loopy belief propagation. Suppose we have a stationary point. Taking $d\mathcal{L}/db(\mathbf{x}_c) = 0$ and $d\mathcal{L}/db(x_i) = 0$ (assuming $n_i \neq 0$) gives the relationships

$$b(\mathbf{x}_c) = \psi(\mathbf{x}_c) \exp(\sum_{i \in c} \lambda_c(x_i) - 1 - \gamma_c), \qquad b(x_i) = \exp(-\frac{1}{n_i} \sum_{c: i \in c} \lambda_c(x_i) - \frac{\gamma_i}{n_i} - 1).$$

Now, we can produce a convergent loopy belief propagation configuration. Suppose the messages are chosen so that $\lambda_c(x_i) = \log \prod_{a \neq c: i \in a} m_{a \to i}(x_i)$ (To see that this is possible, see Eq. 2.19 below). Then the beliefs are

$$\begin{aligned} b(\mathbf{x}_c) &\propto \psi(\mathbf{x}_c) \textstyle\prod_{i \in c} \prod_{a \neq c: i \in a} m_{a \to i}(x_i) &&= \psi(\mathbf{x}_c) \displaystyle\prod_{i \in c} m_{i \to c}(x_i) \\ b(x_i) &\propto \big(\textstyle\prod_{c: i \in c} \prod_{a \neq c: i \in a} m_{a \to i}(x_i) \big)^{-1/n_i} &&= \displaystyle\prod_{c: i \in c} m_{c \to i}(x_i). \end{aligned}$$

The following construction for messages verifies that it is possible to find messages satisfying the criteria for $\lambda_c(x_i)$ above. It can be mechanically checked that $\log \prod_{a \neq c: i \in a} m_{a \to i}(x_i) = \lambda_c(x_i)$.

$$m_{a \to i}(x_i) = \exp\big(-\frac{n_i + 1}{n_i} \lambda_a(x_i) - \frac{1}{n_i} \sum_{c \neq a: i \in c} \lambda_c(x_i) \big). \tag{2.19}$$

To see the converse (that a fixed point of loopy BP gives a stationary point of the Bethe free energy), notice that the above proof can be run "in reverse": Given the messages $m_{a \to i}$, choose $\lambda_c(x_i) = \log \prod_{a \neq c: i \in a} m_{a \to i}(x_i)$, and then observe that for appropriately chosen γ_i and γ_c, this gives a stationary point.

minimizing the KL-divergence from some approximating distribution b to the true distribution p. The essential difference is that mean field restricts b to a simple fully-factorized form, $b(\mathbf{x}) = \prod_i b(x_i)$. Under this large restriction, it is possible to find an exact local minima. Substituting a fully-factorized b into the KL-divergence, when again $p(\mathbf{x}) = \frac{1}{Z}\prod_c \psi(\mathbf{x}_c)$ gives[4]

$$\arg\min_b KL(b||p) = \arg\min_b \sum_i \sum_{x_i} b(x_i)\log b(x_i) - \sum_c \sum_{\mathbf{x}_c} \log\psi(\mathbf{x}_c)\prod_{i\in c} b(x_i). \quad (2.20)$$

Notice that if $p(\mathbf{x})$ is itself fully factorized, it will be possible to set $b(x_i)$ so that $KL(b||p) = 0$, and so mean-field will give exact marginals at the global minimum. On the other hand, if $p(\mathbf{x})$ cannot be well-approximated by a fully factorized distribution, we can expect mean-field to give poor results.

Now, consider updating an individual factor $b(x_j)$ to minimize the KL-divergence with all other factors fixed. It can be shown[5] that the update will be

[4]After substituting, one can drop the constant term corresponding to the partition function.

$$\begin{aligned}\arg\min_b \sum_{\mathbf{x}} b(\mathbf{x})\log\frac{b(\mathbf{x})}{p(\mathbf{x})} &= \arg\min_b \sum_{\mathbf{x}} b(\mathbf{x})\log b(\mathbf{x}) - \sum_{\mathbf{x}} b(\mathbf{x})\sum_c \log\psi(\mathbf{x}_c) \\ &= \arg\min_b \sum_i \sum_{x_i} b(x_i)\log b(x_i) - \sum_c \sum_{\mathbf{x}_c} \log\psi(\mathbf{x}_c)\prod_{i\in c} b(x_i)\end{aligned}$$

[5]Consider the minimization just as a function of $b(x_j)$. Form a Lagrangian enforcing that $\sum_{x_j} b(x_j) = 1$.

$$\mathcal{L}_j = \sum_{x_j} b(x_j)\log b(x_j) - \sum_{c:j\in c}\sum_{\mathbf{x}_c}\log\psi(\mathbf{x}_c)\prod_{i\in c} b(x_i) + \lambda(1 - \sum_{x_j} b(x_j))$$

Taking $d\mathcal{L}_j/db(x_j) = 0$ gives

$$\log b(x_j) + 1 - \sum_{c:j\in c}\sum_{\mathbf{x}_{c\setminus j}}\log\psi(\mathbf{x}_c)\prod_{i\in c:i\neq j} b(x_i) - \lambda = 0.$$

Now, solve for $b(x_j)$.

$$b(x_j) = \exp(\sum_{c:j\in c}\sum_{\mathbf{x}_{c\setminus j}}\log\psi(\mathbf{x}_c)\prod_{i\in c:i\neq j} b(x_i) + \lambda - 1)$$

Figure 2.6: Sets of beliefs.

$$b(x_j) \propto \exp\Big(\sum_{c:j\in c}\sum_{\mathbf{x}_{c\setminus j}} \log\psi(\mathbf{x}_c)\prod_{i\in c:i\neq j} b(x_i)\Big). \tag{2.21}$$

One can either iterate directly using Eq. 2.21 or consider this as a system of equations, and solve it by other means. If iterating, the KL-divergence can never increase, meaning that the updates cannot cycle. However, the mean field objective function (Eq. 2.20) is non-convex in the local beliefs, meaning that different local minima are possible.

Mean-field can be made slightly more powerful by using a "tree-structured" approximating distribution rather than a fully-factorized one. This still allows the KL-divergence to be computed exactly, and gives a strictly more powerful space of approximating distributions.

Wainwright and Jordan [7] give an insightful contrast between mean-field and loopy belief propagation. In mean-field, one exactly minimizes the KL-divergence $\mathrm{KL}(b||p)$, but must drastically restrict the space of approximating distributions. In LBP, meanwhile, local consistency *enlarges* the space of allowed beliefs beyond those that are (globally) consistent, while simultaneously the Bethe approximation means only an approximation of KL-divergence is minimized. Fig. 2.6 visualizes the space of globally consistent beliefs in white. It can by approximated either with a fully-factorized subset (in black), or a locally-consistent superset (in gray).

In the presentation here, mean field was seen as minimizing a function only of univariate beliefs. However, it can also be seen as performing a constrained minimization of a function of both univariate and clique-wise beliefs. In this case a *convex* objective

function is minimized over a *non-convex* constraint set. This leads to a Figure similar to Fig. 2.6, where the set of Fully-Factorized distributions appears nonconvex [7].

Chapter 3

Loss Functions

3.1 Empirical risk minimization

A loss function measures how well a given model fits to some training data. When discussing loss functions, it is useful to remember that the ultimate goal is to fit some aspect of the true distribution, for which the training data is a surrogate. All the loss functions in this thesis can be derived from the perspective of empirical risk minimization. Suppose that the true distribution is $p_0(\mathbf{x})$. Abstractly, learning can be phrased as choosing some distribution p from a set of candidate distributions P. Suppose that we would like to minimize the expected value of some loss function $L(p, \mathbf{x})$. Then, we would like to minimize the "true risk",

$$\arg\min_{p \in P} \sum_{\mathbf{x}} p_0(\mathbf{x}) L(p, \mathbf{x}). \tag{3.1}$$

Of course, this cannot actually be done, since p_0 is unknown. We have access only to some data *sampled* from p_0. In empirical risk minimization, one approximates the true risk with an "empirical risk". If $\hat{X}$ is a set of points $\hat{\mathbf{x}}$ sampled from p_0, one can make a Monte-Carlo approximation

$$\sum_{\mathbf{x}} p_0(\mathbf{x}) L(p, \mathbf{x}) \approx \frac{1}{N} \sum_{\hat{\mathbf{x}} \in \hat{X}} L(p, \hat{\mathbf{x}}), \tag{3.2}$$

where $|\hat{X}| = N$. Minimizing this "empirical risk" is called "empirical risk minimization".

$$\arg\min_{p \in P} \sum_{\hat{\mathbf{x}} \in \hat{X}} L(p, \hat{\mathbf{x}}) \tag{3.3}$$

Note that there is no guarantee that substituting data for the the true distribution like this will yield a consistent estimator. For *fixed* p, the Monte-Carlo approximation in Eq. 3.2 will converge as $N \to \infty$, under mild conditions. However, if P contains infinitely many distributions, this does *not* mean in general that the minimizer of Eq. 3.3 will converge to the minimizer of Eq. 3.1. Statistical learning theory [8] studies the conditions under which this approximation is consistent. Despite their importance, these issues will not be discussed further in this thesis.

Note that when the set of candidate distributions P is large, a direct minimization of Eq. 3.3 will often produce a distribution that fits the training data well, but has a poor true risk. To combat this "overfitting" one will usually add a "regularization" term, which penalizes complex distributions and favors simple ones.

In practice, we will usually not be interested in fitting a joint distribution $p(\mathbf{x})$, but rather a conditional distribution $p(\mathbf{y}|\mathbf{x})$. In this case, the true risk is

$$\arg\min_{p \in P} \sum_{\mathbf{x}} \sum_{\mathbf{y}} p_0(\mathbf{y}, \mathbf{x}) L(p, \mathbf{y}, \mathbf{x}). \tag{3.4}$$

Now, the training data will be some set $\hat{D} = \{(\hat{\mathbf{y}}, \hat{\mathbf{x}})\}$ sampled from p_0. In learning, one again simply selects the distribution with minimum empirical risk.

$$\arg\min_{p \in P} \sum_{(\hat{\mathbf{y}}, \hat{\mathbf{x}}) \in \hat{D}} L(p, \hat{\mathbf{y}}, \hat{\mathbf{x}}), \tag{3.5}$$

Below, the set of candidate distributions (P) and training data ($\hat{X}$ or $\hat{D}$) will be suppressed for notational simplicity.

3.2 The Likelihood and Conditional Likelihood

The (negative-log) likelihood loss is simply the negative log-probability of some data element.

$$L(p, \hat{\mathbf{x}}) = -\log p(\hat{\mathbf{x}}).$$

Usually, the likelihood is defined as $\log p(\hat{\mathbf{x}})$, a quantity to be *maximized* in learning. This thesis uses the negative log probability to maintain consistently that all loss functions should be *minimized*.

To optimize the likelihood is to try to minimize the Kullback-Leibler or KL-divergence between the true distribution, and the one being fit. The KL-divergence is defined by

$$KL(p(\mathbf{x})||q(\mathbf{x})) := \sum_{\mathbf{x}} p(\mathbf{x}) \log \frac{p(\mathbf{x})}{q(\mathbf{x})}.$$

Roughly speaking, the KL-divergence measures how many bits are wasted on average if one builds a code for data coming from p under the *assumption* that the data is coming from q. Importantly, $KL(p||q) = 0$ if and only if $p = q$. See Minka [9] for intuition in the context of graphical models.

Now, suppose the true, unknown distribution is $p_0(\mathbf{x})$. It is easy to see that the true risk is equivalent to the negative log probability.

$$\arg\min_p KL\big(p_0(\mathbf{x})||p(\mathbf{x})\big) \quad = \quad \arg\min_p -\sum_{\mathbf{x}} p_0(\mathbf{x}) \log p(\mathbf{x}).$$

The (negative-log) conditional likelihood loss is very similar.

$$L(p, \hat{\mathbf{y}}, \hat{\mathbf{x}}) = -\log p(\hat{\mathbf{y}}|\hat{\mathbf{x}})$$

Optimizing this turns out to be equivalent to minimizing the *expected* KL-divergence from the true distribution.

$$\begin{aligned}\arg\min_p \sum_{\mathbf{x}} p_0(\mathbf{x}) KL\big(p_0(\mathbf{y}|\mathbf{x})||p(\mathbf{y}|\mathbf{x})\big) &= \arg\min_p -\sum_{\mathbf{x}} p_0(\mathbf{x}) \sum_{\mathbf{y}} p_0(\mathbf{y}|\mathbf{x}) \log p(\mathbf{y}|\mathbf{x}) \\ &= \arg\min_p -\sum_{\mathbf{x}} \sum_{\mathbf{y}} p_0(\mathbf{y}, \mathbf{x}) \log p(\mathbf{y}|\mathbf{x})\end{aligned}$$

Now we consider how to compute the derivative of the conditional likelihood loss. Recall the definition of a CRF.

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_c \psi(\mathbf{y}_c, \mathbf{x})$$

The negative logarithm is

$$L(p, \hat{\mathbf{y}}, \hat{\mathbf{x}}) = -\sum_c \log \psi(\hat{\mathbf{y}}_c, \hat{\mathbf{x}}) + \log Z(\hat{\mathbf{x}}).$$

Now, if the functions ψ are tuned by some parameters $\boldsymbol{\theta}$, the gradient is[1]

$$\frac{\partial L}{\partial \boldsymbol{\theta}} = -\sum_c \frac{\partial}{\partial \boldsymbol{\theta}} \log \psi(\hat{\mathbf{y}}_c, \hat{\mathbf{x}}) + \sum_c \sum_{\mathbf{y}_c} p(\mathbf{y}_c|\hat{\mathbf{x}}) \frac{\partial}{\partial \boldsymbol{\theta}} \log \psi(\mathbf{y}_c, \hat{\mathbf{x}}). \tag{3.6}$$

Notice the meaning of this: one can compute the gradient of the likelihood loss if one can compute the marginals $p(\mathbf{y}_c|\hat{\mathbf{x}})$.

[1]There is a bit of manipulation needed to calculate the derivative of the partition function.

$$\begin{aligned}\frac{\partial}{\partial \boldsymbol{\theta}} \log Z(\mathbf{x}) = \quad & \tfrac{1}{Z(\mathbf{x})} \textstyle\sum_{\mathbf{y}} \frac{\partial}{\partial \boldsymbol{\theta}} \prod_c \psi(\mathbf{y}_c, \mathbf{x}) && = \frac{1}{Z(\mathbf{x})} \sum_{\mathbf{y}} \big(\prod_c \psi(\mathbf{y}_c, \mathbf{x})\big) \sum_c \frac{1}{\psi(\mathbf{y}_c, \mathbf{x})} \frac{\partial}{\partial \boldsymbol{\theta}} \psi(\mathbf{y}_c, \mathbf{x}) \\ = \quad & \textstyle\sum_{\mathbf{y}} p(\mathbf{y}|\mathbf{x}) \sum_c \frac{\partial}{\partial \boldsymbol{\theta}} \log \psi(\mathbf{y}_c, \mathbf{x}) && = \sum_c \sum_{\mathbf{y}_c} p(\mathbf{y}_c|\mathbf{x}) \frac{\partial}{\partial \boldsymbol{\theta}} \log \psi(\mathbf{y}_c, \mathbf{x})\end{aligned}$$

3.3 Approximate Likelihoods, Approximate Inference, and the Exponential Family

(This section, which is based on Wainwright and Jordan [7], can be skipped with minimal loss of continuity.) One fairly common method in practice for parameter fitting with high-treewidth graphical models is to use an algorithm such as loopy belief propagation to compute approximate marginals, and then use these in place of the true marginals to estimate the gradient in Eq. 3.6. This heuristic argument seems to have motivated the original use of this approach. A more principled understanding of this method comes from the perspective of the exponential family. A probability distribution in the exponential family can be defined by

$$p(\mathbf{x};\boldsymbol{\theta}) = \exp(\boldsymbol{\theta}^T\mathbf{f}(\mathbf{x}) - A(\boldsymbol{\theta})),$$

$$A(\boldsymbol{\theta}) = \log \sum_{\mathbf{x}} \exp \boldsymbol{\theta}^T\mathbf{f}(\mathbf{x}),$$

where $\mathbf{f}$ is some vector of "sufficient statistics". (Essentially, the elements of $\mathbf{f}$ can be arbitrary features of $\mathbf{x}$). It can be shown[2] that the first and second order derivatives of A correspond to the expected value, and covariance matrix of $\mathbf{f}$.

$$\frac{d}{d\boldsymbol{\theta}}A(\boldsymbol{\theta}) = \sum_{\mathbf{x}} p(\mathbf{x};\boldsymbol{\theta})f(\mathbf{x}) = E_{\boldsymbol{\theta}}[\mathbf{f}(\mathbf{x})]$$

[2]The computation of the first order derivative is mechanical.

$$\frac{d}{d\boldsymbol{\theta}}A(\boldsymbol{\theta}) = \frac{\sum_{\mathbf{x}} \exp(\boldsymbol{\theta}^T\mathbf{f}(\mathbf{x}))\mathbf{f}(\mathbf{x})}{\sum_{\mathbf{x}} \exp \boldsymbol{\theta}^T\mathbf{f}(\mathbf{x})} = \sum_{\mathbf{x}} p(\mathbf{x};\boldsymbol{\theta})\mathbf{f}(\mathbf{x}) = \boldsymbol{\mu}$$

The second order derivative is somewhat more involved.

$$\begin{aligned}\frac{d^2}{d\boldsymbol{\theta}d\boldsymbol{\theta}^T}A(\boldsymbol{\theta}) = &\quad \sum_{\mathbf{x}} \mathbf{f}(\mathbf{x})\tfrac{d}{d\boldsymbol{\theta}^T}p(\mathbf{x};\boldsymbol{\theta}) &&= \sum_{\mathbf{x}} \mathbf{f}(\mathbf{x})p(\mathbf{x};\boldsymbol{\theta})\left(\mathbf{f}^T(\mathbf{x}) - \boldsymbol{\mu}^T\right)\\ = &\quad \sum_{\mathbf{x}} p(\mathbf{x};\boldsymbol{\theta})\mathbf{f}(\mathbf{x})\mathbf{f}^T(\mathbf{x}) - \boldsymbol{\mu}\boldsymbol{\mu}^T &&= E_{\boldsymbol{\theta}}[(\mathbf{f}(\mathbf{x}) - \boldsymbol{\mu})(\mathbf{f}(\mathbf{x}) - \boldsymbol{\mu})^T]\end{aligned}$$

$$\frac{d^2}{d\boldsymbol{\theta} d\boldsymbol{\theta}^T} A(\boldsymbol{\theta}) = \mathrm{Cov}_{\boldsymbol{\theta}}[\mathbf{f}(\mathbf{x})]$$

Now, consider fitting $\boldsymbol{\theta}$ to some dataset using the likelihood loss. The goal is to minimize

$$\begin{aligned} \frac{1}{N}\sum_{\hat{\mathbf{x}}} L(\hat{\mathbf{x}}) &= -\frac{1}{N}\sum_{\hat{\mathbf{x}}} \log p(\hat{\mathbf{x}}; \boldsymbol{\theta}) \\ &= A(\boldsymbol{\theta}) - \frac{1}{N}\sum_{\hat{\mathbf{x}}} \boldsymbol{\theta}^T \mathbf{f}(\hat{\mathbf{x}}). \end{aligned}$$

Taking the gradient with respect to $\boldsymbol{\theta}$, and setting this to zero gives

$$\sum_{\mathbf{x}} p(\mathbf{x}; \boldsymbol{\theta}) \mathbf{f}(\mathbf{x}) = \frac{1}{N}\sum_{\hat{\mathbf{x}}} \mathbf{f}(\hat{\mathbf{x}}). \tag{3.7}$$

These are called *moment matching* conditions. The expected value of the features under p must be equal to the average value of the features in the data. To illuminate the connection to approximate graphical models and approximate inference, it is useful to rederive this same result by a different route.

Since the matrix of second order derivatives of A is a covariance matrix, it must be positive definite, and so A is convex in $\boldsymbol{\theta}$. Hence, it possible to write A in terms of its conjugate (or Legendre) dual function as

$$A(\boldsymbol{\theta}) = \sup_{\boldsymbol{\mu}} \{\boldsymbol{\theta}^T \boldsymbol{\mu} - A^*(\boldsymbol{\mu})\}. \tag{3.8}$$

The connection to approximate inference and entropy approximations stems from the dual function. It can be shown[3] that A^* is the negative entropy of p, *when the*

[3]By definition, $A^*(\boldsymbol{\mu}) = \sup_{\boldsymbol{\theta}} \{\boldsymbol{\theta}^T \boldsymbol{\mu} - A(\boldsymbol{\theta})\}$.

By taking the derivative of the expression inside the supremum, we see that that if there is a $\boldsymbol{\theta}$ is such that $\boldsymbol{\mu} = E_{\boldsymbol{\theta}}[\mathbf{f}(\mathbf{x})]$, then the supremum will be obtained there. Let $\boldsymbol{\theta}^*$ be the parameters where the maximum is attained. Then, we have that

mean parameters $\boldsymbol{\mu}$ *are achievable.* Let

$$\text{MARG} = \{\boldsymbol{\mu} : \exists \boldsymbol{\theta}, \boldsymbol{\mu} = \sum_{\mathbf{x}} p(\mathbf{x}; \boldsymbol{\theta}) \mathbf{f}(\mathbf{x})\}$$

be the set of achievable mean parameters. The entropy is defined by

$$H(\boldsymbol{\mu}) = -\sum_{\mathbf{x}} p(\mathbf{x}; \boldsymbol{\mu}) \log p(\mathbf{x}; \boldsymbol{\mu}),$$

where $p(\mathbf{x}; \boldsymbol{\mu})$ is a shorthand for the distribution resulting from finding the parameters that yield $\boldsymbol{\mu}$. So, finally,

$$A^*(\boldsymbol{\mu}) = \begin{cases} -H(\boldsymbol{\mu}) & \boldsymbol{\mu} \in \text{MARG} \\ \infty & \boldsymbol{\mu} \notin \text{MARG}. \end{cases} \tag{3.9}$$

Notice, incidentally, that since $A^*(\boldsymbol{\mu})$ is convex $H(\boldsymbol{\mu})$ must be concave. By substituting Eq. 3.9 into Eq. 3.8, we see that we can write the partition function in terms of a constrained optimization.

$$A(\boldsymbol{\theta}) = \sup_{\boldsymbol{\mu} \in \text{MARG}} \{\boldsymbol{\theta}^T \boldsymbol{\mu} + H(\boldsymbol{\mu})\} \tag{3.10}$$

We will need the derivative of A in this variational form. By Danskin's theorem[4], if H is concave, then

$$\begin{aligned} \frac{dA(\boldsymbol{\theta})}{d\boldsymbol{\theta}} &= \frac{d}{d\boldsymbol{\theta}} \sup_{\boldsymbol{\mu} \in \text{MARG}} \{\boldsymbol{\theta}^T \boldsymbol{\mu} + H(\boldsymbol{\mu})\} \\ &= \arg \max_{\boldsymbol{\mu} \in \text{MARG}} \{\boldsymbol{\theta}^T \boldsymbol{\mu} + H(\boldsymbol{\mu})\}. \end{aligned} \tag{3.11}$$

$$\boldsymbol{\theta}^{*T} \boldsymbol{\mu} - A(\boldsymbol{\theta}^*) = \sum_{\mathbf{x}} p(\mathbf{x}; \boldsymbol{\theta}^*) \big(\boldsymbol{\theta}^{*T} \mathbf{f}(\mathbf{x}) - A(\boldsymbol{\theta}^*)\big) = \sum_{\mathbf{x}} p(\mathbf{x}; \boldsymbol{\theta}^*) \log p(\mathbf{x}; \boldsymbol{\theta}^*).$$

If there does not exist such a $\boldsymbol{\theta}$, then $A^*(\boldsymbol{\mu}) = \infty$.

[4]Danskin's theorem states generally that if $f(\mathbf{x}, \mathbf{z})$ is convex in $\mathbf{x}$ for all $\mathbf{z}$, then $g(\mathbf{x}) = \max_{\mathbf{z}} f(\mathbf{x}, \mathbf{z})$ is convex in $\mathbf{x}$, and $\frac{d}{d\mathbf{x}} g(\mathbf{x}) = \frac{d}{d\mathbf{x}} f(\mathbf{x}, \bar{\mathbf{z}})$, where $\bar{\mathbf{z}} = \arg\max_{\mathbf{z}} f(\mathbf{x}, \mathbf{z})$.

Eq. 3.10 can be used to give a variational representation of the likelihood loss.

$$\begin{aligned} L(\hat{\mathbf{x}}) &= -\log p(\hat{\mathbf{x}}; \boldsymbol{\theta}) \\ &= \sup_{\boldsymbol{\mu} \in \text{MARG}} \{\boldsymbol{\theta}^T \boldsymbol{\mu} + H(\boldsymbol{\mu})\} - \boldsymbol{\theta}^T \mathbf{f}(\hat{\mathbf{x}}) \end{aligned}$$

Using Eq. 3.11 we can also calculate the gradient of this loss.

$$\frac{dL(\hat{\mathbf{x}})}{d\boldsymbol{\theta}} = \arg \max_{\boldsymbol{\mu} \in \text{MARG}} \{\boldsymbol{\theta}^T \boldsymbol{\mu} + H(\boldsymbol{\mu})\} - \mathbf{f}(\hat{\mathbf{x}}) \tag{3.12}$$

So, we see that the three problems of performing inference, computing likelihoods or likelihood gradients, and computing the partition function all face essentially the same computational difficulty, namely performing the optimization in Eq. 3.10.

If $\boldsymbol{\theta}$ is fit to optimize the likelihood loss on data, the solution will satisfy

$$\frac{1}{N} \sum_{\hat{\mathbf{x}}} \frac{dL(\hat{\mathbf{x}})}{d\boldsymbol{\theta}} = \mathbf{0},$$

which, substituting Eq. 3.12, can be solved to yield

$$\arg \max_{\boldsymbol{\mu} \in \text{MARG}} \{\boldsymbol{\theta}^T \boldsymbol{\mu} + H(\boldsymbol{\mu})\} = \frac{1}{N} \sum_{\hat{\mathbf{x}}} f(\hat{\mathbf{x}}).$$

These are the same moment matching constraints from Eq. 3.7. The reason for deriving this alternative form, as we will see below, is that we can understand the effect of approximations.

Now, we observe that optimization developed previously for exact inference in graphical models (Eq. 2.16) can be seen as a special case of Eq. 3.10. To see this, identify the vector $\boldsymbol{\theta}$ with the values $-\log \psi(\mathbf{x}_c)$, and set as "features" indicator functions for all clique configurations. That is, use the features $f_i(\mathbf{x}) = \delta(\mathbf{x}_c = \mathbf{a})$ for all possible c and $\mathbf{a}$.

Of course, just casting the problem in terms of the exponential family does not cause any of the previous computational problems to disappear. In general graphs, MARG will be difficult to characterize. Hence, one typically simplifies this, e.g. using local consistency. Similarly, the entropy is in general intractable to compute (and not even defined for $\boldsymbol{\mu} \notin \text{MARG}$), meaning it must also be approximated. Take some approximate entropy $\tilde{H}$, and an approximate marginal polytope $\tilde{\text{MARG}}$. It is natural to define an approximate partition function

$$\tilde{A}(\boldsymbol{\theta}) = \sup_{\boldsymbol{\mu} \in \tilde{\text{MARG}}} \{\boldsymbol{\theta}^T \boldsymbol{\mu} + \tilde{H}(\boldsymbol{\mu})\}, \tag{3.13}$$

yielding the approximate moments

$$\tilde{\boldsymbol{\mu}} = \frac{d\tilde{A}}{d\boldsymbol{\theta}} = \arg \max_{\boldsymbol{\mu} \in \tilde{\text{MARG}}} \{\boldsymbol{\theta}^T \boldsymbol{\mu} + \tilde{H}(\boldsymbol{\mu})\}, \tag{3.14}$$

and an approximate likelihood

$$\tilde{L}(\hat{\mathbf{x}}) = \sup_{\boldsymbol{\mu} \in \tilde{\text{MARG}}} \{\boldsymbol{\theta}^T \boldsymbol{\mu} + \tilde{H}(\boldsymbol{\mu})\} - \boldsymbol{\theta}^T \mathbf{f}(\hat{\mathbf{x}}).$$

The derivative of the approximate likelihood is particularly interesting. The derivative of the true likelihood (Eq. 3.12) is the difference of the moments and the features of one data element. Now we have

$$\frac{d\tilde{L}(\hat{\mathbf{x}})}{d\boldsymbol{\theta}} = \arg \max_{\boldsymbol{\mu} \in \tilde{\text{MARG}}} \{\boldsymbol{\theta}^T \boldsymbol{\mu} + \tilde{H}(\boldsymbol{\mu})\} - \mathbf{f}(\hat{\mathbf{x}}),$$

i.e. the difference of the *approximate* moments and the features. If $\boldsymbol{\theta}$ is fit to optimize this approximate likelihood on data, then it will satisfy

$$\arg \max_{\boldsymbol{\mu} \in \tilde{\text{MARG}}} \{\boldsymbol{\theta}^T \boldsymbol{\mu} + \tilde{H}(\boldsymbol{\mu})\} = \frac{1}{N} \sum_{\hat{\mathbf{x}}} f(\hat{\mathbf{x}}).$$

Thus, this procedure is a kind of "approximate moment matching". The approximate moments are matched to data, rather than the true moments.

Now, let us make the connection to graphical models more explicit. Consider performing inference in a graphical model with the Bethe approximation to the entropy, and the local consistency approximation to the marginal polytope. Then, we have the correspondences

$$\begin{aligned}
\boldsymbol{\theta} &\leftrightarrow \{\log \psi(\mathbf{x}_c)\} \cup \{\log \psi(x_i)\} \\
\boldsymbol{\mu} &\leftrightarrow \{b(\mathbf{x}_c)\} \cup \{b(x_i)\} \\
\hat{H}(\boldsymbol{\mu}) &\leftrightarrow -\sum_c \sum_{\mathbf{x}_c} b(\mathbf{x}_c) \log b(\mathbf{x}_c) - n_i \sum_i \sum_{x_i} b(x_i) \log b(x_i) \\
\tilde{\text{MARG}} &\leftrightarrow \{b : \sum_{\mathbf{x}_c} b(\mathbf{x}_c) = 1, \sum_{x_i} b(\mathbf{x}_i) = 1, \\
&\qquad b(\mathbf{x}_c) \geq 0, b(x_i) \geq 0, b(x_i) = \sum_{\mathbf{x}_{c \setminus i}} b(\mathbf{x}_c)\}
\end{aligned} \tag{3.15}$$

and so the maximization $\max_{\boldsymbol{\mu} \in \tilde{\text{MARG}}} \{\boldsymbol{\theta}^T \boldsymbol{\mu} + \tilde{H}(\boldsymbol{\mu})\}$ is equivalent to the optimization

$$\max \sum_c \sum_{\mathbf{x}_c} b(\mathbf{x}_c) \log \psi(\mathbf{x}_c) - \sum_c \sum_{\mathbf{x}_c} b(\mathbf{x}_c) \log b(\mathbf{x}_c) - \sum_i n_i \sum_{x_i} b(x_i) \log b(x_i)$$

subject to the constraints in Eq. 3.15. This is precisely the optimization we previously saw in Section 2.2.4 (phrased as a maximization instead of a minimization).

Henceforth the "approximate likelihood loss" will refer to the objective function

$$L(p, \hat{\mathbf{x}}) = -\sum_c \log \psi(\hat{\mathbf{x}}_c) + \log \tilde{Z}. \tag{3.16}$$

The approximate log-partition function is

$$\log \tilde{Z} = \sum_c \sum_{\mathbf{x}_c} \bar{b}(\mathbf{x}_c) \log \psi(\mathbf{x}_c) - \tilde{H}(\bar{b}),$$

where $\tilde{H}(b)$ is some approximate entropy, and $\bar{b}$ is the set of beliefs resulting from inference. Now let $\boldsymbol{\phi}$ the parameters of the factors. We know from Eq. 3.14 that

$$\frac{d \log \tilde{Z}}{d \log \psi(\mathbf{x}_c)} = \bar{b}(\mathbf{x}_c),$$

and so

$$\begin{aligned} \frac{d \log \tilde{Z}}{d\boldsymbol{\phi}} &= \sum_c \sum_{\mathbf{x}_c} \frac{d \log \tilde{Z}}{d \log \psi(\mathbf{x}_c)} \frac{d \log \psi(\mathbf{x}_c)}{d\boldsymbol{\phi}} \\ &= \sum_c \sum_{\mathbf{x}_c} \bar{b}(\mathbf{x}_c) \frac{d \log \psi(\mathbf{x}_c)}{d\boldsymbol{\phi}}. \end{aligned}$$

Hence, the approximate likelihood loss has the derivatives alluded to above: the same derivatives as for the true likelihood, but where approximate marginals are used in place of the true marginals.

$$\frac{\partial L}{\partial \boldsymbol{\phi}} = -\sum_c \frac{\partial}{\partial \boldsymbol{\phi}} \log \psi(\hat{\mathbf{x}}) + \sum_c \sum_{\mathbf{x}_c} \bar{b}(\mathbf{x}_c) \frac{\partial}{\partial \boldsymbol{\phi}} \log \psi(\mathbf{x}_c). \tag{3.17}$$

Similarly, we can define an "approximate conditional likelihood loss",

$$L(p, \hat{\mathbf{y}}, \hat{\mathbf{x}}) = -\sum_c \log \psi(\hat{\mathbf{y}}_c, \hat{\mathbf{x}}) + \log \tilde{Z}(\hat{\mathbf{x}}), \tag{3.18}$$

$$\log \tilde{Z}(\hat{\mathbf{x}}) = \sum_c \sum_{\mathbf{x}_c} \bar{b}(\mathbf{x}_c) \log \psi(\mathbf{x}_c) - H(b),$$

which has the gradient

$$\frac{\partial L}{\partial \boldsymbol{\phi}} = -\sum_c \frac{\partial}{\partial \boldsymbol{\phi}} \log \psi(\hat{\mathbf{y}}_c, \hat{\mathbf{x}}) + \sum_c \sum_{\mathbf{y}_c} \bar{b}(\mathbf{y}_c) \frac{\partial}{\partial \boldsymbol{\phi}} \log \psi(\mathbf{y}_c, \hat{\mathbf{x}}). \tag{3.19}$$

3.4 Pseudolikelihood

The pseudolikelihood [10, 11] loss is

$$L(p, \hat{\mathbf{x}}) = -\sum_i \log p(\hat{x}_i | \hat{\mathbf{x}}_{\neq i}).$$

The usual justification for this is that the pseudolikelihood is *consistent*[5]. That is, if $p_0 \in P$, then

$$p_0 = \arg\min_{p \in P} \sum_{\mathbf{x}} p_0(\mathbf{x}) L(p, \mathbf{x}).$$

Hence if the model is well-specified, as the amount of data increases we can expect minimization of the pseudolikelihood to converge to the true distribution.

However, if $p_0 \notin P$ (e.g. because the graphical model asserts conditional in dependencies that do not hold on p_0) then the pseudolikelihood may give poor results. The author is not aware of any argument that in the case of an misspecified model, the pseudolikelihood will converge to an optimal estimate in any useful sense.

If estimating a conditional distribution, the conditional pseudolikelihood is

$$L(p, \hat{\mathbf{y}}, \hat{\mathbf{x}}) = -\sum_i \log p(\hat{y}_i | \hat{\mathbf{y}}_{\neq i}, \hat{\mathbf{x}}). \tag{3.20}$$

[5]To see this, set up a Lagrangian enforcing that all conditional distributions are normalized.

$$\mathcal{L} = \sum_{\mathbf{x}} p_0(\mathbf{x}) L(p, \mathbf{x}) + \sum_i \sum_{\mathbf{x}_{\neq i}} \lambda(\mathbf{x}_{\neq i})(1 - \sum_{x_i} p(x_i | \mathbf{x}_{\neq i}))$$

Taking $d\mathcal{L}/dp(x_i|\mathbf{x}_{\neq i}) = 0$, one obtains $-p_0(\mathbf{x})/p(x_i|\mathbf{x}_{\neq i}) - \lambda(\mathbf{x}_{\neq i}) = 0$, or, equivalently, $p(x_i|\mathbf{x}_{\neq i}) \propto p_0(\mathbf{x})$. This gives $p(x_i|\mathbf{x}_{\neq i}) = p_0(x_i|\mathbf{x}_{\neq i})$. If this is true for all i, then $p = p_0$.

Similar arguments hold for the conditional likelihood: Given a correct model, it gives a consistent estimator. Given an incorrect model, it does not give an optimal estimate for any natural definition of "optimal". Liang and Jordan [12] give an asymptotic analysis of the likelihood, pseudolikelihood, and conditional likelihood.

3.5 Univariate Loss Functions

What are we trying to do in learning? The best results will be achieved if the model is chosen to give the best performance in whatever way it will be used.

Purposive learning is most valuable when the model is *misspecified*. In some simple cases, e.g. flipping a biased coin, we can say with high confidence that the uncertainty can be modeled with a specific form of distribution [13], meaning we have a well-specified model. In most all real applications, however, this is not the case, and the true distribution p_0 is best regarded to some degree as an unknown "black box" [14]. The standard reason for using the likelihood is not that we truly wish to minimize KL-divergence, but that we want to *drive it to zero*, i.e. find the true distribution. If we do not assume that $p_0 \in P$, this justification for the likelihood cannot be used.

The idea of adapting the learning procedure to the specific application is so general that one can say little in the abstract. Still, we can consider the basic question: what inference algorithm will we run on the model? Suppose that the application calls for using a marginalization algorithm to compute $p(y_i|\mathbf{x})$. If that is the case, no aspect of the joint distribution $p(\mathbf{y}|\mathbf{x})$ other than the marginals will ever be observed. The idea of univariate loss functions is to fit the model only to predict univariate marginals well. If one can "trade" joint accuracy for marginal accuracy, such an approach makes sense.

3.5.1 Univariate Conditional Likelihood

The univariate conditional likelihood loss is

$$L(p, \hat{\mathbf{y}}, \hat{\mathbf{x}}) = -\sum_i \log p(\hat{y}_i|\hat{\mathbf{x}}).$$

If one is only interested in marginal accuracy, an obvious idea would be to minimize the expected sum of univariate KL-divergences. This results in the univariate conditional likelihood.

$$\begin{aligned}
\arg\min_p \sum_{\mathbf{x}} p_0(\mathbf{x}) \sum_i KL\big(p_0(y_i|\mathbf{x})||p(y_i|\mathbf{x})\big) &= \arg\min_p -\sum_{\mathbf{x}} p_0(\mathbf{x}) \sum_i \sum_{y_i} p_0(y_i|\mathbf{x}) \log p(y_i|\mathbf{x}) \\
&= \arg\min_p -\sum_{\mathbf{x}} p_0(\mathbf{x}) \sum_i \sum_{\mathbf{y}} p_0(\mathbf{y}|\mathbf{x}) \log p(y_i|\mathbf{x}) \\
&= \arg\min_p -\sum_{\mathbf{x}} \sum_{\mathbf{y}} p_0(\mathbf{y}, \mathbf{x}) \sum_i \log p(y_i|\mathbf{x})
\end{aligned}$$

The univariate conditional likelihood was proposed by Kakade et al. [15], who also provide an algorithm for calculating the gradient for models with exact inference and linear features.

3.5.2 Univariate Quadratic Loss

The univariate conditional quadratic loss[16] is

$$L(p, \hat{\mathbf{y}}, \hat{\mathbf{x}}) = \sum_i \big(-2p(\hat{y}_i|\hat{\mathbf{x}}) + \sum_{y_i} p(y_i|\hat{\mathbf{x}})^2\big).$$

This loss stems from trying to minimize the expected squared difference of marginal probabilities.

$$
\begin{aligned}
&\arg\min_p \sum_{\mathbf{x}} p_0(\mathbf{x}) \sum_i \sum_{y_i} \big(p_0(y_i|\mathbf{x}) - p(y_i|\mathbf{x})\big)^2 \\
&= \arg\min_p \sum_{\mathbf{x}} p_0(\mathbf{x}) \sum_i \big(\sum_{y_i} -2p_0(y_i|\mathbf{x})p(y_i|\mathbf{x}) + \sum_{y_i'} p(y_i'|\mathbf{x})^2\big) \\
&= \arg\min_p \sum_{\mathbf{x}} p_0(\mathbf{x}) \sum_i \sum_{\mathbf{y}} p_0(\mathbf{y}|\mathbf{x}) \big(-2p(y_i|\mathbf{x}) + \sum_{y_i'} p(y_i'|\mathbf{x})^2\big) \\
&= \arg\min_p \sum_{\mathbf{x}} \sum_{\mathbf{y}} p_0(\mathbf{y}, \mathbf{x}) \sum_i \big(-2p(y_i|\mathbf{x}) + \sum_{y_i'} p(y_i'|\mathbf{x})^2\big)
\end{aligned}
$$

3.5.3 Smoothed Univariate Classification Loss

The smoothed classification loss is

$$
L(p, \hat{\mathbf{y}}, \hat{\mathbf{x}}) = \sum_i \sigma\big(\lambda(\max_{y_i \neq \hat{y}_i} p(y_i|\hat{\mathbf{x}}) - p(\hat{y}_i|\hat{\mathbf{x}}))\big),
$$

where λ is a control parameter, and σ is a smooth "sigmoid" approximation to the step function, e.g. $\sigma(a) = 1/(1 + \exp(-a))$.

In a common situation, given the observation $\mathbf{x}$ one needs to produce a single "guess" $\mathbf{y}^*$ of the hidden variables. The most common way to do this is MAP inference.

$$
\mathbf{y}^* = \arg\max_{\mathbf{y}} p(\mathbf{y}|\mathbf{x}) \tag{3.21}
$$

Though this has great intuitive appeal, there are many circumstances where this is not the best guess to make. To use MAP inference is to maximize the probability that the entire vector $\mathbf{y}$ is exactly equal to the true vector. In cases where $\mathbf{y}$ is uncertain and high dimensional, $\max_{\mathbf{y}} p(\mathbf{y}|\mathbf{x})$ will often be an extremely small number. We often do not care about guessing a vector that has a 0.001% chance of being exactly correct rather than a 0.0009% chance.

An alternative is to guess the vector $\mathbf{y}^*$ that has the maximum *number of variables*

correct.

$$y_i^* = \arg\max_{y_i} p(y_i|\mathbf{x}) \tag{3.22}$$

The process of computing the marginals, and then using Eq. 3.22 is called Maximum Posterior Marginal (MPM) inference (Section 2.2).

The smoothed univariate classification loss tries to fit p so that MPM inference gives the best possible results. In principle, this would dictate

$$\arg\min_p \sum_{\mathbf{y}} \sum_x p_0(\mathbf{y}, \mathbf{x}) \sum_i \delta[y_i \neq \arg\max_{y_i} p(y_i|\mathbf{x})]$$

for an indicator function[6] δ. However, notice that for fixed $\mathbf{x}$ and $\mathbf{y}$, $\delta[y_i \neq \arg\max_{y_i} p(y_i|\mathbf{x})]$ is non-differentiable with respect to p. Hence, Gross et al. [17] suggested approximating the indicator function with a smooth sigmoid function.

$$\arg\min_p \sum_{\mathbf{y}} \sum_x p_0(\mathbf{y}, \mathbf{x}) \sum_i \sigma\left(\lambda(\max_{y_i' \neq y_i} p(y_i'|\mathbf{x}) - p(y_i|\mathbf{x}))\right)$$

The parameter λ determines how closely the sigmoid approximates the step function. For larger, λ, the approximation is better, but the loss function is more nonlinear, which seems to increase the prevalence of local minima. Gross et al. also provide an algorithm for calculating the gradient for models with exact inference and linear features.

3.6 Clique Loss Functions.

Univariate loss functions are not consistent. If $p_0 \in P$, selecting $p = p_0$ will yield perfect marginals, and so no other distribution can have a strictly better true risk. However, one can construct cases where some other distribution achieves a loss equal

[6]The indicator function is defined by $\delta[\text{expr}] = 1$ if expr is true, and $\delta[\text{expr}] = 0$ if expr is false.

to that of p_0. (Imagine a complex joint distribution that has simple marginals like $p(y_i|\mathbf{x}) = \text{const.}$)

We can define loss functions analogous to those above that target clique accuracy, rather than univariate accuracy. These functions are consistent.

The consistency of clique-wise loss functions stems from the fact that an MRF can be seen as a member of the exponential family with indicator functions on cliques as sufficient statistics. (Section 3.3) If all clique-wise marginals match, the distributions thus must be the same. The same holds for a CRF: after conditioning, a CRF is just and MRF, and hence if the conditional marginals $p(\mathbf{y}_c|\mathbf{x})$ are always equal to $p_0(\mathbf{y}_c|\mathbf{x})$, then $p = p_0$. Note, however, that this argument does not apply to the Clique Classification Error.

3.6.1 Clique Conditional Likelihood

$$L(p, \hat{\mathbf{y}}, \hat{\mathbf{x}}) = -\sum_c \log p(\hat{\mathbf{y}}_c|\hat{\mathbf{x}})$$

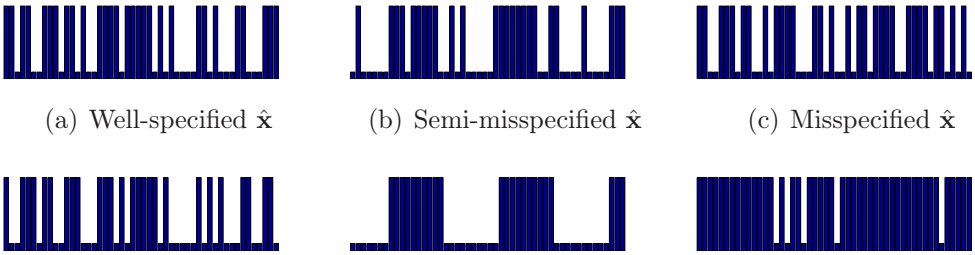
3.6.2 Clique Quadratic Loss

$$L(p, \hat{\mathbf{y}}, \hat{\mathbf{x}}) = \sum_c \big(-2p(\hat{\mathbf{y}}_c|\hat{\mathbf{x}}) + \sum_{\mathbf{y}_c} p(\mathbf{y}_c|\hat{\mathbf{x}})^2\big)$$

3.6.3 Smoothed Clique Classification Error

If each clique configuration is predicted independently, the smoothed clique classification error tries to measure how many cliques are predicted incorrectly. Note that it is not enforced that a variable takes a single value in the different cliques. So it will not be possible in general to produce vectors $\mathbf{y}^*$ that achieve the clique classification error.

$$L(p, \hat{\mathbf{y}}, \hat{\mathbf{x}}) = \sum_c \sigma\big(\lambda(\max_{\mathbf{y}_c \neq \hat{\mathbf{y}}_c} p(\mathbf{y}_c|\hat{\mathbf{x}}) - p(\hat{\mathbf{y}}_c|\hat{\mathbf{x}}))\big)$$



(a) Well-specified $\hat{\mathbf{x}}$ (b) Semi-misspecified $\hat{\mathbf{x}}$ (c) Misspecified $\hat{\mathbf{x}}$

(d) Well-specified $\hat{\mathbf{y}}$ (e) Semi-misspecified $\hat{\mathbf{y}}$ (f) Misspecified $\hat{\mathbf{y}}$

Figure 3.1: Example tractable data.

3.7 Experiments on Chain Graphs

This section presents experiments learning a simple (tractable) binary chain graph model with all of the above loss functions. There are three different learning data sets (Figure 3.1).

- **Well-specified.** In the first, $\hat{\mathbf{x}}$ is generated randomly from a uniform distribution, and then $\hat{\mathbf{y}}$ is created using MCMC with randomly selected CRF parameters.
- **Semi-misspecified.** In the second, $\hat{\mathbf{y}}$ is generated by creating sequences of alternating 0s and 1s, each in groups of ten. The "noisy" input $\hat{\mathbf{x}}$ is made by taking $\hat{\mathbf{y}}$, and setting 60% of the variables to random values.
- **Misspecified.** In the third, $\hat{\mathbf{y}}$ is set to be all one random value, with five randomly chosen variables changed. The "noisy" input $\hat{\mathbf{x}}$ is made by taking $\hat{\mathbf{y}}$, and setting 75% of the variables to random values.

For each data set, a CRF was trained with a variety of loss functions. Since the conditional likelihood is convex, the results of learning on it were used to initialize all other loss functions.

Abbreviation	Loss
pseudo	Conditional Pseudolikelihood
cl	Conditional Likelihood
ucl	Univariate Conditional Likelihood
ccl	Clique Conditional Likelihood
uquad	Univariate Quadratic
cquad	Clique Quadratic
uclass	Smoothed Univariate Classification Error
cclass	Clique Classification Error
s.uclass	Smoothed Univariate Classification Error
s.cclass	Smoothed Clique Classification Error

Table 3.1: Loss function abbreviations.

The goal of these experiments is to compare all the above loss functions, testing how important is the assumption of being well-specified. Figures 3.2 and 3.3 show the results averaged over 100 training sets, evaluated by univariate classification accuracy. Notice that for well-specified training data, all loss functions ultimately give approximately the same performance. However, for misspecified data, the conditional likelihood asymptotically performs worse that other loss functions, and the pseudo-likelihood performs worse still. Figures 3.4-3.9 show the same experiment, with test evaluation on various loss functions.

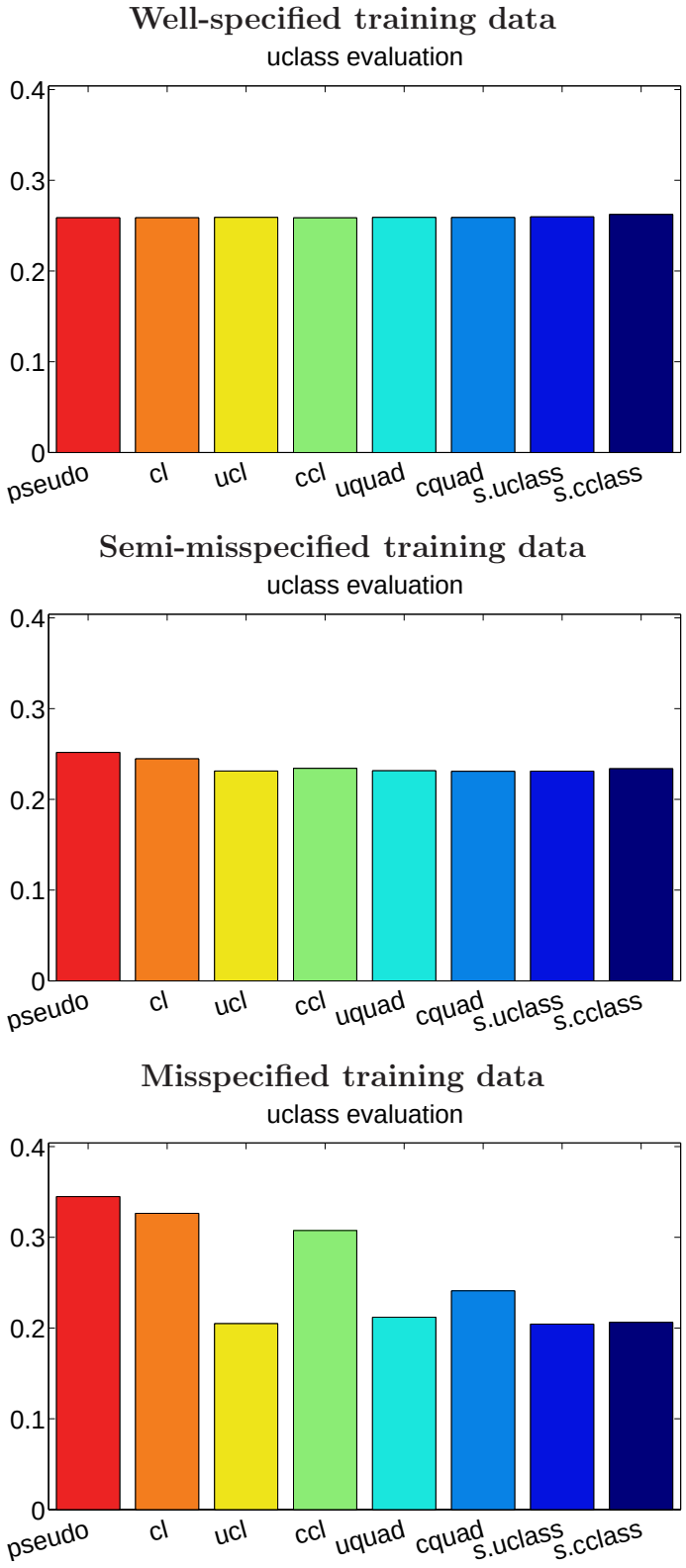


Figure 3.2: Univariate classification errors on test data for a CRF trained with various loss functions on three different data sets. For well-specified data, all loss functions perform comparably, while for misspecified data, there are large asymptotic differences.

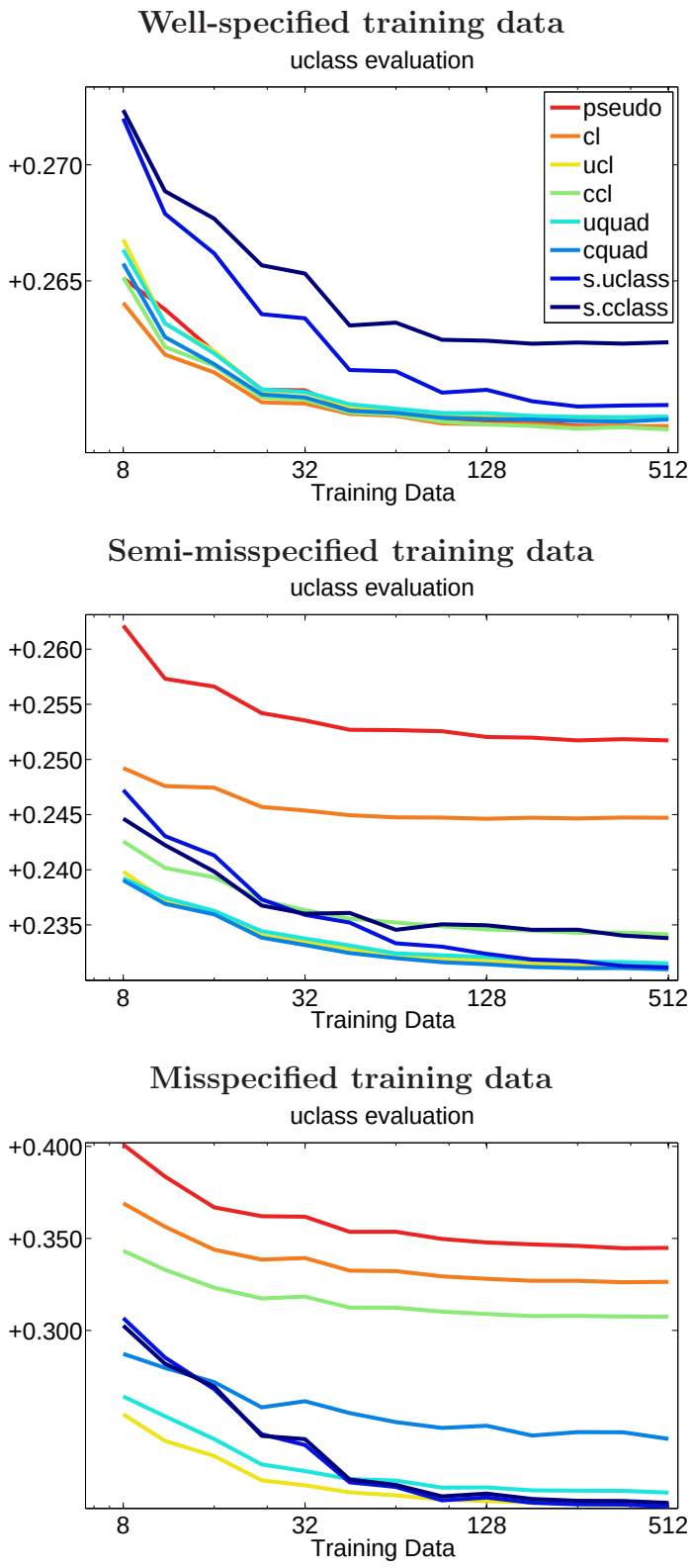


Figure 3.3: Univariate classification errors on test data for a CRF trained with various loss functions on three different data sets. For well-specified data, all loss functions perform comparably, while for misspecified data, there are large asymptotic differences.

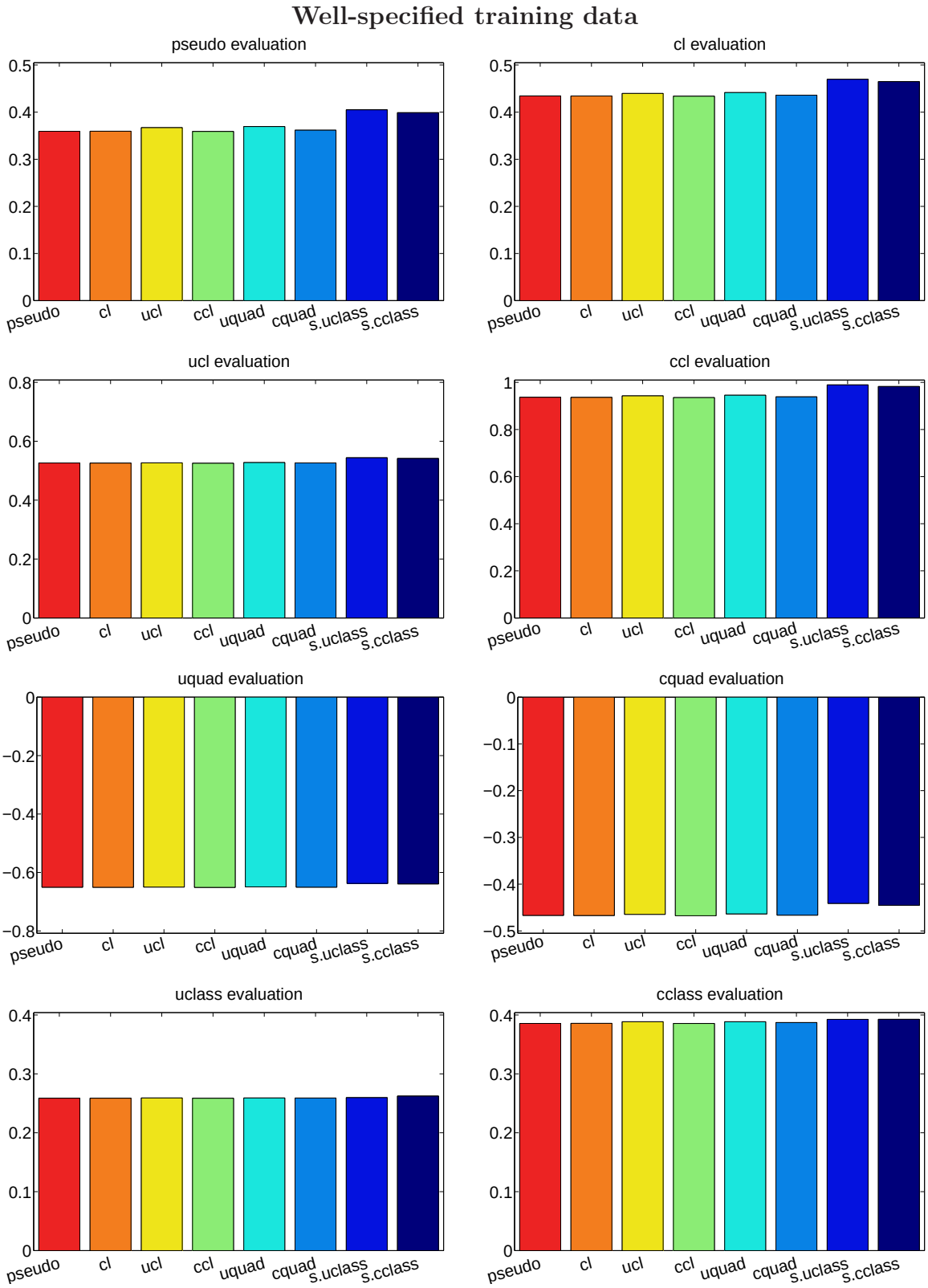


Figure 3.4: Evaluations of different loss functions on a well-specified CRF. Roughly speaking, if data is plentiful, the loss function used for training is unimportant.

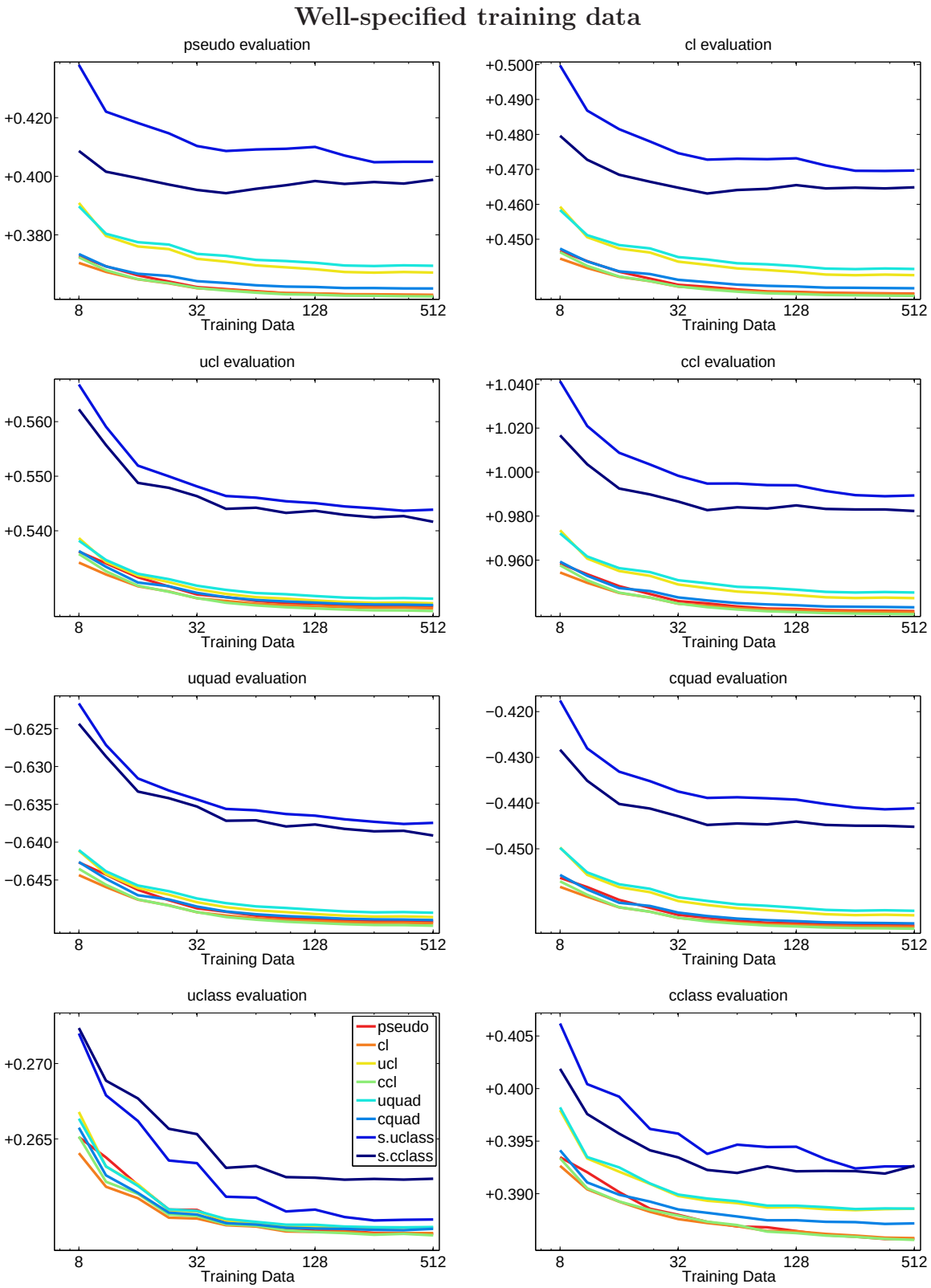


Figure 3.5: Evaluations of different loss functions on a well-specified CRF. Roughly speaking, if data is plentiful, the loss function used for training is unimportant.

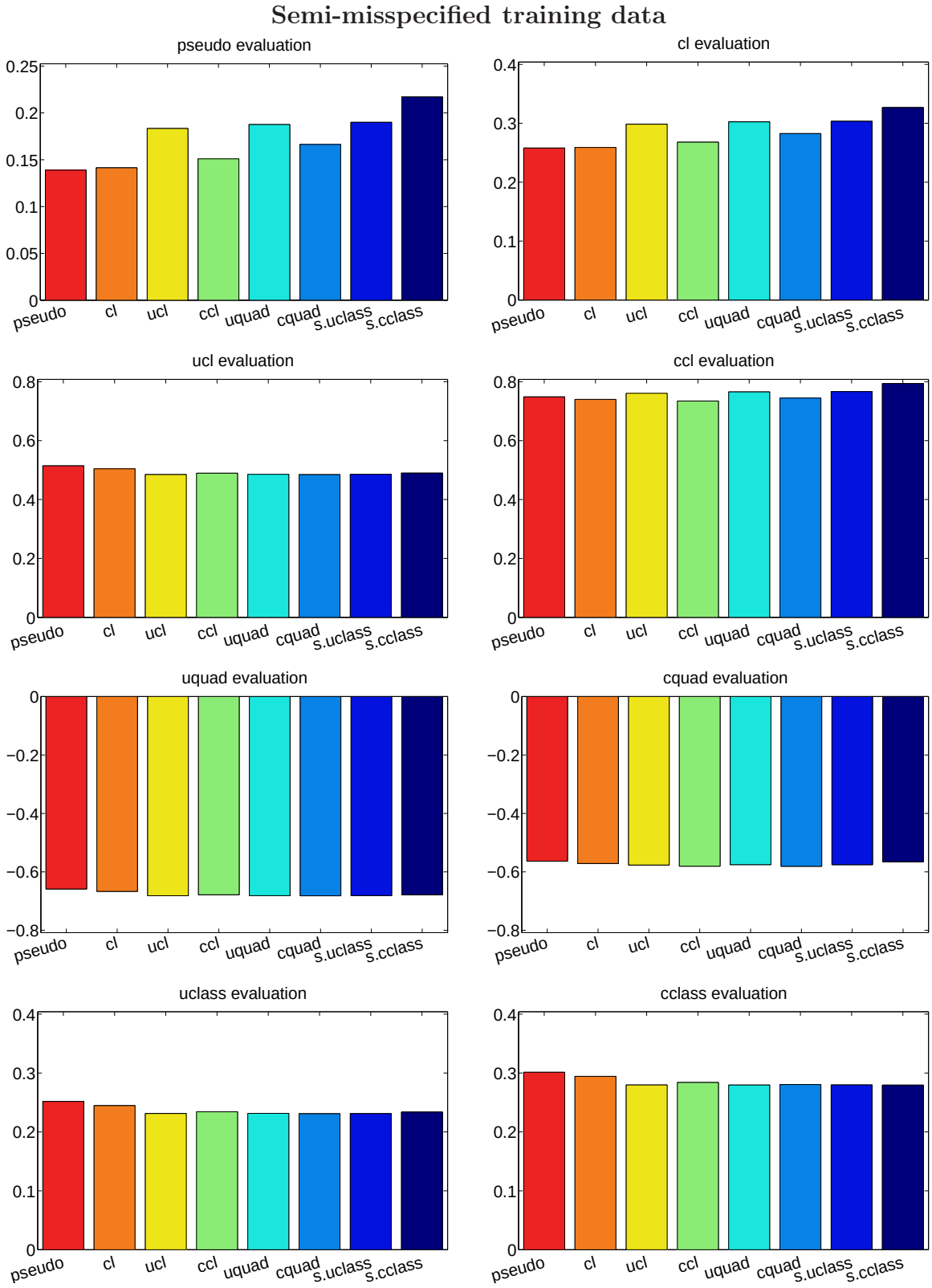


Figure 3.6: Evaluations of different loss functions on a semi-misspecified CRF. There are asymptotic differences between the different loss functions, though they are not large.

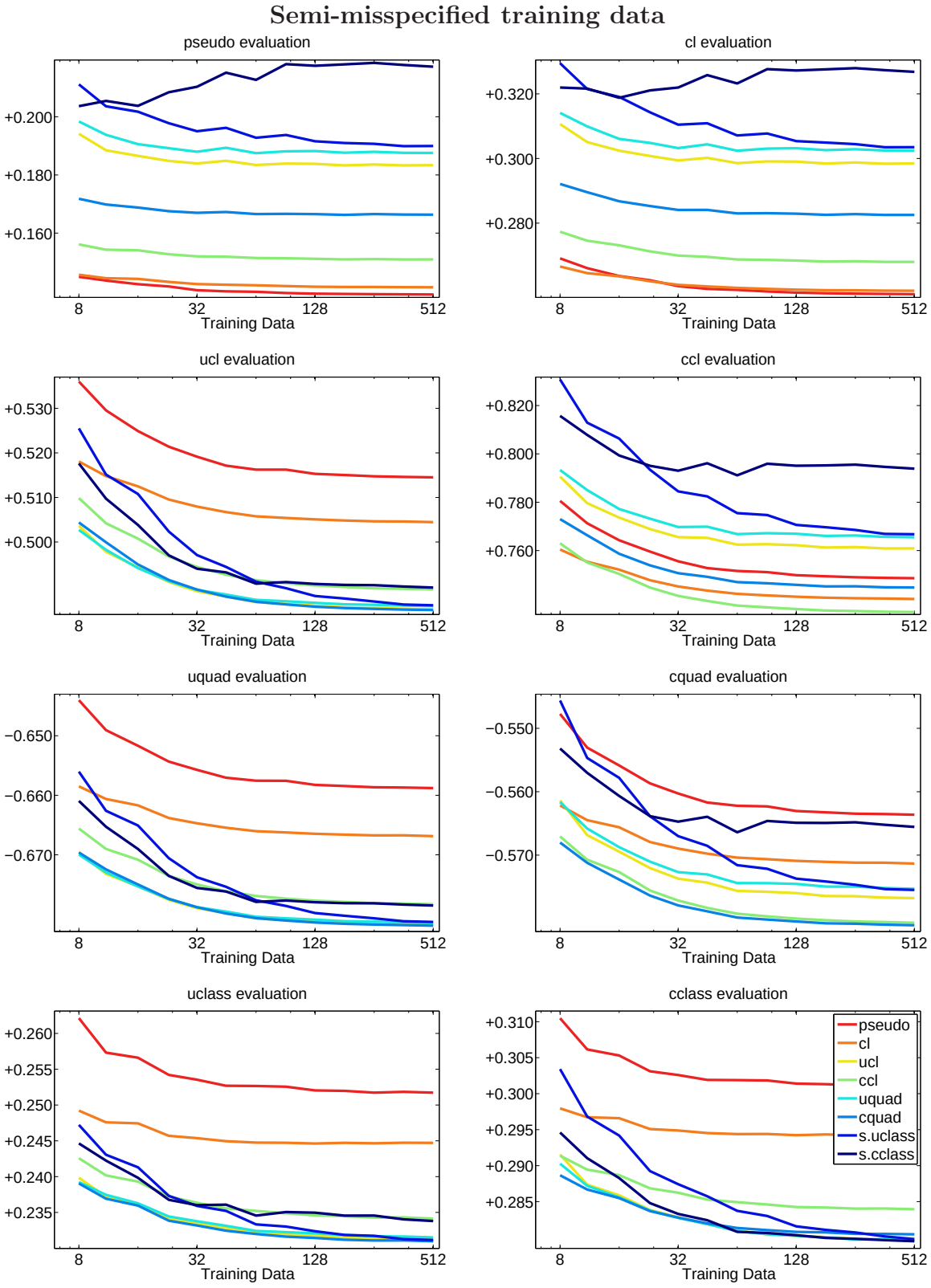


Figure 3.7: Evaluations of different loss functions on a semi-misspecified CRF. There are asymptotic differences between the different loss functions, though they are not large.

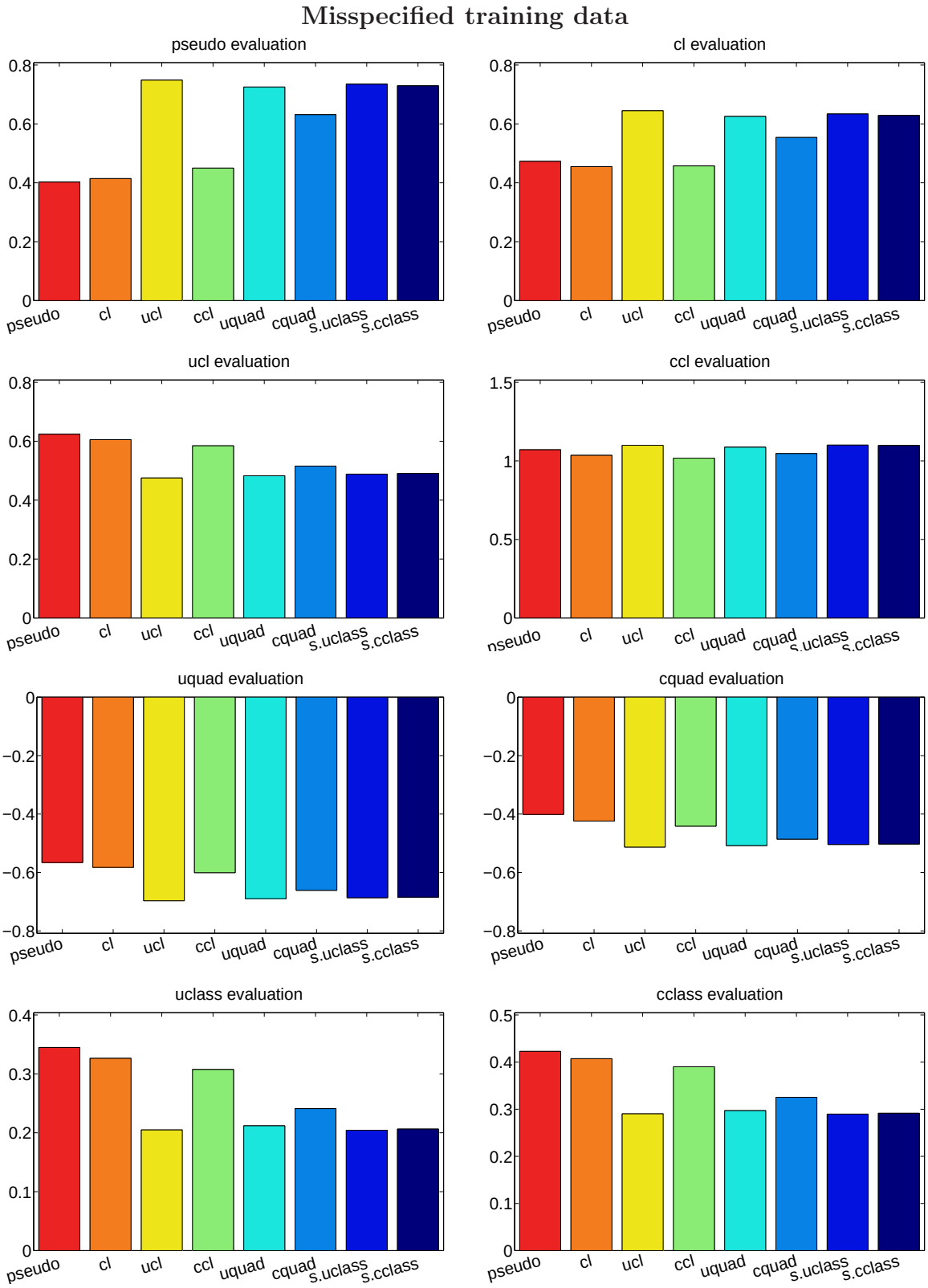


Figure 3.8: Evaluations of different loss functions on a misspecified CRF. There are large asymptotic differences between the different loss functions.

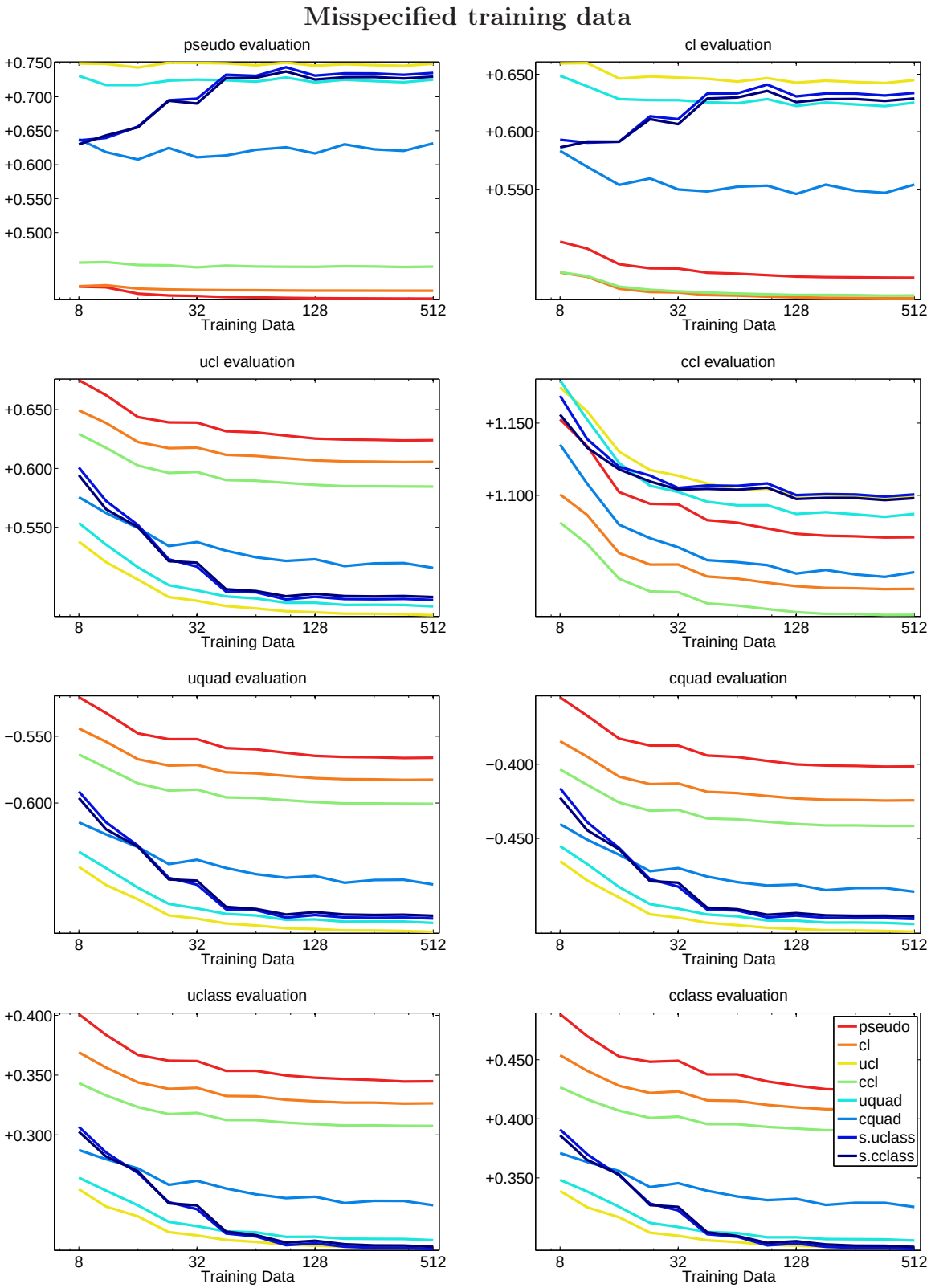


Figure 3.9: Evaluations of different loss functions on a misspecified CRF. There are large asymptotic differences between the different loss functions.

Chapter 4

Implicit Fitting

4.1 Overview

The basic idea of this chapter is to consider a graphical model as defining an energy function which, when minimized, gives predicted marginals. When learning, the goal is to shape this energy function so that the minima give accurate beliefs.

Suppose one would like to fit a CRF using any of the univariate or clique-wise loss functions from Chapter 3. Thus, $p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_c \psi(\mathbf{y}_c, \mathbf{x})$, and one would like to fit ψ so as to minimize some loss

$$\sum_{(\hat{\mathbf{y}}, \hat{\mathbf{x}})} L(p, \hat{\mathbf{y}}, \hat{\mathbf{x}}).$$

In general graphs, this will be extremely difficult to do. All the proposed loss functions (except the pseudolikelihood) require computing univariate or clique-wise marginals, which is intractable. Thus, we cannot even evaluate the loss , much less adjust the model to minimize it.

This chapter takes a different approach. Rather than fitting *the distribution that the CRF represents*, fit *the beliefs that the CRF produces under approximate inference.* There are two major advantages to this. First, learning becomes tractable: one only

needs approximate beliefs to compute a loss. We will see below that the gradient of the loss is also computable. Secondly, this is a more *purposive* form of learning. The model is fit to give the best possible predictions, taking into account all approximations that must be made in the inference step.

As a simple example, if one were to fit using the univariate conditional likelihood, the traditional loss would be (Sec. 3.5.1)

$$L(p, \hat{\mathbf{y}}, \hat{\mathbf{x}}) = -\sum_i \log p(\hat{y}_i|\hat{\mathbf{x}}).$$

The strategy envisioned here, meanwhile would fit

$$L(\psi, \hat{\mathbf{y}}, \hat{\mathbf{x}}) = -\sum_i \log b_\psi(\hat{y}_i|\hat{\mathbf{x}}),$$

where b_ψ are the beliefs that result from performing inference on the model defined by ψ.

One difficulty here is that learning is *implicit*: The model is fit purely in terms of the minima of the inference energy function. A technical difficulty is calculating the derivative of the predicted beliefs with respect to each parameter of the model. Formally, suppose that ψ is parametrized by some vector $\boldsymbol{\theta}$. Then, we would like to calculate changes in local beliefs, e.g.

$$\frac{db_\psi(y_i|\mathbf{x})}{d\theta_j} \quad \text{or} \quad \frac{db_\psi(\mathbf{y}_c|\mathbf{x})}{d\theta_j}.$$

We find that this can be done using a strategy of *implicit differentiation*, as long as the inference energy function is convex.

4.2 Marginal Inference as an Optimization

We saw in Section 2.2.4 that loopy belief propagation attempts to perform the optimization

$$\text{minimize} \sum_c \sum_{\mathbf{y}_c} b(\mathbf{y}_c) \log b(\mathbf{y}_c) + \sum_i n_i \sum_{y_i} b(y_i) \log b(y_i) - \sum_c \sum_{\mathbf{y}_c} b(\mathbf{y}_c) \log \psi(\mathbf{y}_c, \mathbf{x}).$$

subject to the constraints

$$\begin{aligned} b(y_i) \geq 0 \qquad & b(\mathbf{y}_c) \geq 0 \\ \sum_{y_i} b(y_i) = 1 \qquad & \sum_{\mathbf{y}_c} b(\mathbf{y}_c) = 1 \\ b(y_i) = \sum_{\mathbf{y_c} \setminus i} b(\mathbf{y}_c) \quad . & \end{aligned}$$

Here, we generalize this somewhat, to the form

$$\begin{aligned} \text{minimize} \quad & \textstyle\sum_c \sum_{\mathbf{y}_c} w(\mathbf{y}_c, \mathbf{x}) b(\mathbf{y}_c) \log b(\mathbf{y}_c) \quad + \sum_i \sum_{y_i} w(y_i, \mathbf{x}) b(y_i) \log b(y_i) \\ & \textstyle + \sum_c \sum_{\mathbf{y}_c} v(\mathbf{y}_c, \mathbf{x}) b(\mathbf{y}_c) \qquad + \sum_i \sum_{y_i} v(y_i, \mathbf{x}) b(y_i), \end{aligned}$$

subject to the same constraints. The final terms involving $v(y_i, \mathbf{x})b(y_i)$ are linearly dependent on the terms $v(\mathbf{y}_c, \mathbf{x})b(\mathbf{y}_c)$, but are included for convenience.

This optimization is different from a standard Bethe optimization in several ways:

1. The log factors $\log \psi(\mathbf{y}_c, \mathbf{x})$ are replaced with functions $v(\mathbf{y}_c, \mathbf{x})$.

2. Instead of fixed constants (1 or n_i) for the entropy terms, these are now full functions w.

3. The entropy terms w can depend on the input $\mathbf{x}$.

This is equivalent to an optimization of the form

$$\begin{aligned} \text{minimize} \quad & F(\mathbf{b}) && = \mathbf{w}(\mathbf{x})^T(\mathbf{b} \odot \log \mathbf{b}) + \mathbf{v}(\mathbf{x})^T\mathbf{b} \\ \text{s.t.} \quad & A\mathbf{b} && = \mathbf{d} \\ & \mathbf{b} && \geq \mathbf{0}, \end{aligned} \tag{4.1}$$

where $\odot$ is the elementwise product. Here, $\mathbf{b} = \{b(y_i)\} \cup \{b(\mathbf{y}_c)\}$ is a vector of all univariate and clique-wise beliefs. Similarly, $\mathbf{w}(\mathbf{x}) = \{w(y_i, \mathbf{x})\} \cup \{w(\mathbf{y}_c, \mathbf{x})\}$, and $\mathbf{v}(\mathbf{x}) = \{v(y_i, \mathbf{x})\} \cup \{v(\mathbf{y}_c, \mathbf{x})\}$. A boldface ($\mathbf{b}$, $\mathbf{w}$, or $\mathbf{v}$) will always be used to specify these elements in the form of vectors, while a standard font (b, w, or v) will be used for the traditional representation as functions. Using both forms of notation simplifies the presentation below significantly.

4.3 Optimization of the Dual

This section describes an algorithm for optimization of problems in the form of Eq. 4.1. The advantages over standard message passing algorithms are simplicity, generality, and convergence guarantees. Generality is meant to indicate that the algorithm applies to any (convex) problem of the form of Eq. 4.1. This allows additional flexibility in fitting models, as we will see in later sections. The disadvantages include the larger number of iterations required in practice, and the poorer computational scaling with respect to the number of variables.

The algorithm is a straightforward use of Newton's method on the Lagrange dual of Eq. 4.1. First, take the Lagrangian. As we will see below it is not necessary to explicitly enforce that $\mathbf{b} \geq 0$.

$$\mathcal{L} = \mathbf{w}^T(\mathbf{b} \odot \log \mathbf{b}) + \mathbf{v}^T\mathbf{b} + \boldsymbol{\lambda}^T(A\mathbf{b} - \mathbf{d})$$

Convex duality theory states that if Eq. 4.1 is convex (i.e. if $\mathbf{w} > 0$), then if we find some $\boldsymbol{\lambda}$ and $\mathbf{b}$ such that $d\mathcal{L}/d\mathbf{b} = \mathbf{0}$ and $d\mathcal{L}/d\boldsymbol{\lambda} = \mathbf{0}$, then $\mathbf{b}$ will be an optimum of the original problem. Taking $d\mathcal{L}/d\mathbf{b} = \mathbf{0}$, and solving for $\mathbf{b}$ gives[1]

$$\mathbf{b}(\boldsymbol{\lambda}) = \exp\big(-(\mathbf{v} + A^T\boldsymbol{\lambda}) \oslash \mathbf{w} - 1\big),$$

where $\oslash$ denotes elementwise division. Notice that $\mathbf{b}(\boldsymbol{\lambda}) > 0$. Hence, we are left with the (unconstrained) Lagrange dual problem $\text{maximize}_{\boldsymbol{\lambda}}\ g(\boldsymbol{\lambda})$, where the Lagrange dual function $g(\boldsymbol{\lambda})$ simplifies[2] into

$$g(\boldsymbol{\lambda}) = -\mathbf{w}^T\mathbf{b}(\boldsymbol{\lambda}) - \boldsymbol{\lambda}^T\mathbf{d}. \tag{4.2}$$

To complete Newton's method, we require the gradient and Hessian of g. These also have simple forms.

$$\begin{aligned} \frac{dg}{d\boldsymbol{\lambda}} &= A\mathbf{b}(\boldsymbol{\lambda}) - \mathbf{d} \\ \frac{d^2g}{d\boldsymbol{\lambda}d\boldsymbol{\lambda}^T} &= -A\ \text{diag}\big(\mathbf{b}(\boldsymbol{\lambda}) \oslash \mathbf{w}\big)A^T \end{aligned}$$

Since the sparse Hessian is available, using Newton's method, as suggested above, is natural. However, solving large sparse linear systems scales super-linearly in the number of variables (when using a direct solver), so this can be costly in large prob-

[1] $d\mathcal{L}/d\mathbf{b} = \mathbf{w} \odot (1 + \log \mathbf{b}) + \mathbf{v} + A^T\boldsymbol{\lambda} = \mathbf{0}$. Hence, $\log \mathbf{b} = -(\mathbf{v} + A^T\boldsymbol{\lambda}) \oslash \mathbf{w} - 1$.

[2] By definition,

$$g(\boldsymbol{\lambda}) := \min_{\mathbf{b}} \mathcal{L}(\mathbf{b}, \boldsymbol{\lambda}) = \mathbf{w}^T(\mathbf{b}(\boldsymbol{\lambda}) \odot \log \mathbf{b}(\boldsymbol{\lambda})) + \mathbf{v}^T\mathbf{b}(\boldsymbol{\lambda}) + \boldsymbol{\lambda}^T(A\mathbf{b}(\boldsymbol{\lambda}) - \mathbf{d}).$$

Expanding $\log \mathbf{b}(\boldsymbol{\lambda})$ and canceling terms gives Eq. 4.2.

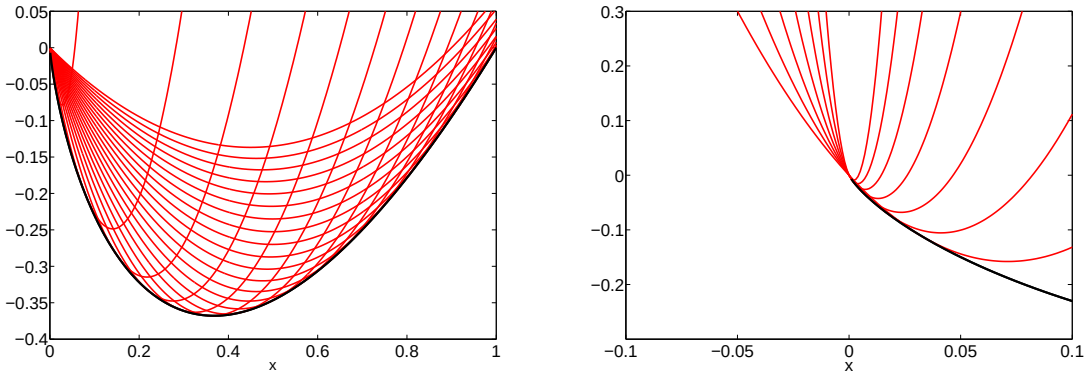


Figure 4.1: Bounds on entropy. Black: $x \log x$. Red: $x(\log x^0 - 1) + \frac{x^2}{x^0}$ for various x^0.

lems. One can alternatively disregard the Hessian, and use methods that scale better, e.g. L-BFGS or nonlinear conjugate gradient search. All practical experience is that the primal algorithm below is more efficient. However, it is possible that a more careful analysis of the above problem could yield a more efficient algorithm.

4.4 Optimization of the Primal

This section describes a novel successive approximation scheme for optimizing Eq. 4.1 in the primal. This method is based on a parametrizable global upper bound on $x \log x$. The bound is, for any x and x^0 greater than zero,

$$x \log x \leq x(\log x^0 - 1) + \frac{x^2}{x^0}.$$

This is pictured in Fig. 4.1 for a range of different x^0. Equality occurs when $x = x^0$. Note that a second-order Taylor approximation would be $x \log x \approx -\frac{1}{2}x^0 + x \log x^0 + (\frac{x^2}{2x^0})$, which provides a better local approximation but does not give a bound.

The basic idea of this method is to repeatedly make the above approximation, resulting in a quadratic optimization problem under linear constraints. After each subproblem is solved, the bound is tightened. Hypothetically, this could be done

with the following algorithm[3].

1. Repeat until convergence:

   (a) $\mathbf{g} \leftarrow \mathbf{v} + \mathbf{w} \odot (\log \mathbf{b}^0 - 1)$

   (b) $H \leftarrow 2\text{diag}(\mathbf{w} \oslash \mathbf{b}^0)$

   (c) Perform the optimization

$$\begin{aligned} \text{minimize} \quad F(\mathbf{b}) &= \mathbf{g}^T\mathbf{b} + \frac{1}{2}\mathbf{b}^T H \mathbf{b} \\ \text{s.t.} \quad A\mathbf{b} &= \mathbf{d} \\ \mathbf{b} &\geq \mathbf{0} \end{aligned}$$

   (d) $\mathbf{b}_0 \leftarrow \mathbf{b}$

This works fine, but performing each sub-optimization is rather expensive. If not for the presence of the constraint $\mathbf{b} \geq 0$, this would be a quadratic optimization under linear constraints, and so could be solved efficiently, with out iteration (see below).

Here, a different strategy is pursued. The basic idea is to use the above quadratic approximation simultaneously as a bound on the entropy, and also as a *penalty function*, enforcing that the beliefs are positive. This is possible because progressively smaller b^0 place increasing penalty on beliefs where $b < 0$. So, we proceed by minimizing the quadratic system, disregarding the positivity constraint. For terms where $b_i > 0$, simply set b_i^0 equal to b_i. For terms where $b_i < 0$, "shrink" b_i^0 towards 0 using a sequence ϵ decreasing towards zero. This algorithm is summarized as follows.

[3]Here, $\mathbf{b} \log \mathbf{b}$ is approximated by

$$(\log \mathbf{b}^0 - 1)^T\mathbf{b} + \mathbf{b}^T\text{diag}(\mathbf{1} \oslash \mathbf{b}^0)\mathbf{b}.$$

Hence $\mathbf{w}^T(\mathbf{b} \odot \log \mathbf{b})$ is approximated by

$$(\mathbf{w} \odot (\log \mathbf{b}^0 - 1))^T\mathbf{b} + \frac{1}{2}\mathbf{b}^T(2\text{diag}(\mathbf{w} \oslash \mathbf{b}^0))\mathbf{b}.$$

1. Repeat until convergence:

   (a) $\mathbf{g} \leftarrow \mathbf{v} + \mathbf{w} \odot (\log \mathbf{b}^0 - 1)$

   (b) $H \leftarrow 2\text{diag}(\mathbf{w} \oslash \mathbf{b}^0)$

   (c) Perform the optimization

$$\begin{aligned} \text{minimize} \quad F(\mathbf{b}) &= \mathbf{g}^T\mathbf{b} + \frac{1}{2}\mathbf{b}^T H\mathbf{b} \\ \text{s.t.} \quad A\mathbf{b} &= \mathbf{d}. \end{aligned}$$

   (d) For all i,

      i. If $b_i > 0$ then $b_i^0 \leftarrow b_i$.

      ii. Otherwise $b_i^0 \leftarrow \epsilon(c_i)$ and $c_i \leftarrow c_i + 1$.

Step (d) may, at first glance, seem unnecessarily complex. One might, instead, let beliefs exponentially decay towards zero, replacing step (d)ii. with something like $b_i^0 \leftarrow b_i^0/10$. Unfortunately, this can lead to cycling. Using a decreasing sequence ϵ guarantees that the penalty for violating $b_i < 0$ increases over time, even if there are occasionally iterations where $b_i > 0$. The experiments below use the sequence $\epsilon(c) = 1/(10c)^2$.

It is well known that a convex quadratic optimization problem under linear constraints can be reduced to a single least-squares problem [18, section 10.4.2]. Briefly, the known result[4] is that if

[4]This is fairly easy to show. Suppose that $\mathbf{x}^* = \arg\min \mathbf{g}^T\mathbf{x} + \frac{1}{2}\mathbf{x}^T H\mathbf{x}$ s.t. $A\mathbf{x} = \mathbf{d}$, where, H is assumed to be symmetric positive definite. Then, the Lagrangian would be

$$\mathcal{L} = \mathbf{g}^T\mathbf{x} + \frac{1}{2}\mathbf{x}^T H\mathbf{x} + \boldsymbol{\lambda}^T(A\mathbf{x} - \mathbf{d}).$$

By taking the derivatives $d\mathcal{L}/d\mathbf{x} = \mathbf{0}$ and $d\mathcal{L}/d\boldsymbol{\lambda} = 0$, we obtain the conditions $\mathbf{g} + H\mathbf{x} + A^T\boldsymbol{\lambda} = 0$, and $A\mathbf{x} - \mathbf{d} = 0$. One can solve the first equation to obtain

$$\mathbf{x}^* = -H^{-1}(\mathbf{g} + A^T\boldsymbol{\lambda}). \tag{4.3}$$

Substituting this into the second equation results in $-AH^{-1}(\mathbf{g} + A^T\boldsymbol{\lambda}) = \mathbf{d}$, which can be solved

Algorithm 1 Primal Belief Optimization Algorithm

1. Initialize $\mathbf{b}$.
2. For all i, $c_i \leftarrow 1$
3. Repeat until convergence:
   (a) $\mathbf{g} \leftarrow \mathbf{w} \odot (\log \mathbf{b} - 1) + \mathbf{v}$
   (b) $H^{-1} \leftarrow \frac{1}{2}\mathrm{diag}\big(\mathbf{b} \oslash \mathbf{w}\big)$
   (c) $\lambda \leftarrow -\big(AH^{-1}A^T\big)^{-1}\big(\mathbf{d} + AH^{-1}\mathbf{g}\big)$
   (d) $\mathbf{b}' \leftarrow -H^{-1}(\mathbf{g} + A^T\boldsymbol{\lambda})$.
   (e) For all i,
      i. If $b'_i > 0$, then $b_i \leftarrow b'_i$.
      ii. Otherwise, $b_i \leftarrow \epsilon(c_i)$ and $c_i \leftarrow c_i + 1$.

$$\mathbf{b}^* = \arg\min \mathbf{g}^T\mathbf{b} + \frac{1}{2}\mathbf{b}^T H\mathbf{b} \text{ subject to } A\mathbf{b} = \mathbf{d},$$

then one can obtain $\mathbf{b}^*$ by first solving

$$\boldsymbol{\lambda} = -\big(AH^{-1}A^T\big)^{-1}\big(\mathbf{d} + AH^{-1}\mathbf{g}\big),$$

and then substituting the result to solve

$$\mathbf{b}^* = -H^{-1}(\mathbf{g} + A^T\boldsymbol{\lambda}).$$

The entire method is summarized as Algorithm 1. Note that the cost of this algorithm is totally dominated by solving the sparse linear system in step 1(c).

The differences of this method to a constrained Newton's method are, explicitly,

1. A local *upper bound* is used, rather than a second-order Taylor expansion.

for

$$\boldsymbol{\lambda} = -\big(AH^{-1}A^T\big)^{-1}\big(\mathbf{d} + AH^{-1}\mathbf{g}\big). \tag{4.4}$$

2. The constraint $\mathbf{b} \geq \mathbf{0}$ is enforced only asymptotically.

3. Line searches are not necessary.

The upper bounding strategy used here is very robust, and deals with the constraint $\mathbf{b} \geq \mathbf{0}$. However, unlike Newton's method, it does not have quadratic convergence guarantees. It may be possible to have the best of both methods by using the approach suggested here for early iterations, and switching to Newton's method in later iterations.

In large problems– e.g. grids of size 100x100 or larger– it becomes expensive to solve the sparse linear system in Step 2(c) of Algorithm 1. This can be avoided by a block optimization. All variables in smaller regions are optimized over, with the rest held constant. Let A_k and $\mathbf{d}_k$ correspond to the constraints relevant to block k of variables. (Thus, A_k takes the columns of A indexed by k, and all rows that are not identically zero on those columns, while $\mathbf{d}_k$ is the entries of $\mathbf{d}$ for those rows.) The principal difference is that when optimizing over block k, instead of constraining $A_k\mathbf{b}_k = \mathbf{d}_k$, one needs to account for the other variables. Thus, one instead constrains it to be equal to $\mathbf{d}_k - A_{-k}\mathbf{b}_{-k}$, where A_{-k} is the matrix taking all variables not in k as input and giving their output on the constraints relevant to block k. (Put another way, A_{-k} is "the rest" of the rows of A when A_k is removed). Experimentally, using 55x55 pixel regions with 5 pixel overlap between neighboring blocks works well with binary variables.

Fig. 4.2 shows an example of convergence, visualizing univariate beliefs. For more details, see Section 6.5.

4.5 Implicit Differentiation

This section will begin with a review of implicit differentiation in general before discussing the application to the graphical models. Consider a relationship between

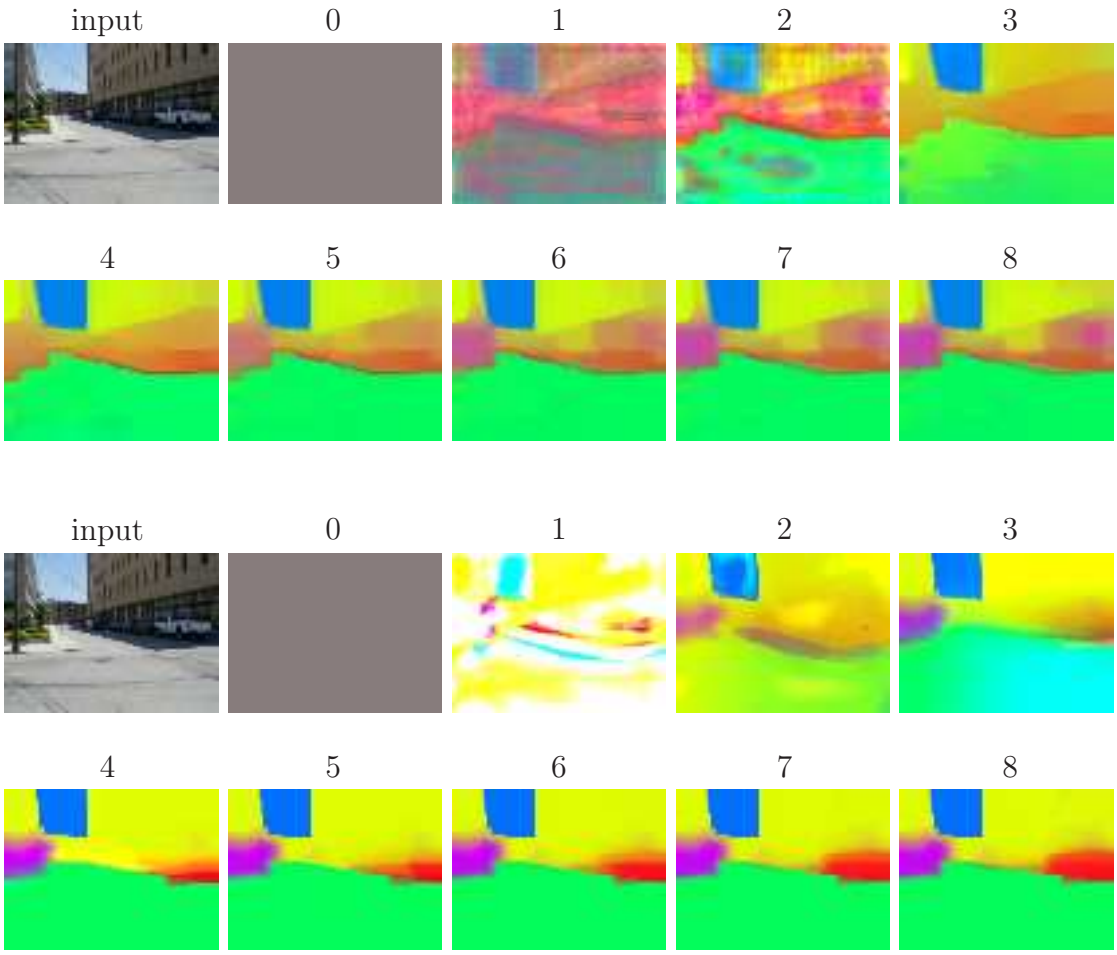


Figure 4.2: Convergence of the primal belief optimization algorithm. Top: Fixed entropy. Bottom: Fit entropy.

Algorithm 2 Block Primal Belief Optimization Algorithm

1. Initialize $\mathbf{b}$.
2. For all i, $c_i \leftarrow 1$
3. Repeat until convergence:
   (a) For all blocks k:
      i. $\mathbf{g} \leftarrow \mathbf{w}_k \odot (\log \mathbf{b}_k - 1) + \mathbf{v}_k$
      ii. $H^{-1} \leftarrow \frac{1}{2}\mathrm{diag}\big(\mathbf{b}_k \oslash \mathbf{w}_k\big)$
      iii. $\lambda \leftarrow -\big(A_k H^{-1} A_k^T\big)^{-1}\big(\mathbf{d}_k - A_{-k}\mathbf{b}_{-k} + A_k H^{-1}\mathbf{g}\big)$
      iv. $\mathbf{b}'_k \leftarrow -H^{-1}(\mathbf{g} + A_k^T \boldsymbol{\lambda})$.
      v. For all $i \in k$,
         A. If $b'_i > 0$, then $b_i \leftarrow b'_i$.
         B. Otherwise, $b_i \leftarrow \epsilon(c_i)$ and $c_i \leftarrow c_i + 1$.

two groups of variables, implicitly defined by

$$\mathbf{f}(\mathbf{x}, \mathbf{y}) = \mathbf{0}.$$

If $\mathbf{x}$ and $\mathbf{y}$ are two points satisfying this relationship, it can be shown[5] that the derivative of $\mathbf{y}$ with respect to $\mathbf{x}$ will be

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}^T} = -\big(\frac{\partial \mathbf{f}}{\partial \mathbf{y}^T}\big)^{-1} \frac{\partial \mathbf{f}}{\partial \mathbf{x}^T}. \tag{4.5}$$

Here, we only need the simpler result where $\mathbf{x}$ is scalar.

[5]Imagine some small perturbation $\delta\mathbf{x}$, resulting in a corresponding perturbation $\delta\mathbf{y}$. These must satisfy

$$\mathbf{f}(\mathbf{x} + \delta\mathbf{x}, \mathbf{y} + \delta\mathbf{y}) = \mathbf{0},$$

which can be approximated to first order by

$$\mathbf{f}(\mathbf{x}, \mathbf{y}) + \frac{\partial \mathbf{f}(\mathbf{x}, \mathbf{y})}{\partial \mathbf{x}^T}\delta\mathbf{x} + \frac{\partial \mathbf{f}(\mathbf{x}, \mathbf{y})}{\partial \mathbf{y}^T}\delta\mathbf{y} = \mathbf{0}.$$

Substituting $\mathbf{f}(\mathbf{x}, \mathbf{y}) = \mathbf{0}$, this can be solved to give

$$\delta\mathbf{y} = -\big(\frac{\partial \mathbf{f}}{\partial \mathbf{y}^T}\big)^{-1} \frac{\partial \mathbf{f}}{\partial \mathbf{x}^T}\delta\mathbf{x}.$$

$$\frac{\partial \mathbf{y}}{\partial x} = -\big(\frac{\partial \mathbf{f}}{\partial \mathbf{y}^T}\big)^{-1} \frac{\partial \mathbf{f}}{\partial x}. \tag{4.6}$$

The relevance here is the implicit relationship given by minimizing the inference energy function. Let $\boldsymbol{\theta}$ denote the parameters of the energy function. *If there were no constraints*, the relationship would be

$$\frac{\partial F(\mathbf{b}, \boldsymbol{\theta})}{\partial \mathbf{b}} = \mathbf{0},$$

and so, substituting $\partial F / \partial \mathbf{b}$ for $\mathbf{f}$ in Eq. 4.6,we would have the derivatives

$$\frac{\partial \mathbf{b}}{\partial \theta} = -\big(\frac{\partial^2 F}{\partial \mathbf{b} \partial \mathbf{b}^T}\big)^{-1} \frac{\partial^2 F}{\partial \mathbf{b} \partial \theta}.$$

If we are to include the equality constraints that $A\mathbf{b} = \mathbf{d}$, a more complex solution is necessary, though based on the same ideas. The result can be summarized by the following.

Claim: Let $F(\mathbf{b}, \theta)$ be strictly convex over $\mathbf{b}$. If $\mathbf{b}(\theta) = \arg\min_{\mathbf{b}} F(\mathbf{b}, \theta)$ such that $A\mathbf{b} = \mathbf{d}$, then

$$\frac{d\mathbf{b}}{d\theta} = \big(D^{-1}A^T(AD^{-1}A^T)^{-1}AD^{-1} - D^{-1}\big)\frac{\partial^2 F}{\partial \mathbf{b} \partial \theta},$$

where $D := \frac{\partial^2 F}{\partial \mathbf{b} \partial \mathbf{b}^T}$.

See the Appendix of this chapter for a proof.

4.6 Learning

By the vector chain rule, the derivative of the loss with respect to a given parameter can be decomposed into the gradient of the loss with respect to the beliefs and the derivative of the beliefs with respect to the parameter. Substituting the result from the previous section for the latter, we have

$$\begin{aligned}\frac{dL}{d\theta} &= \frac{dL}{d\mathbf{b}^T}\frac{d\mathbf{b}}{d\theta} \\ &= \frac{dL}{d\mathbf{b}^T}\big(D^{-1}A^T(AD^{-1}A^T)^{-1}AD^{-1} - D^{-1}\big)\frac{\partial^2 F}{\partial \mathbf{b}\partial\theta}. \end{aligned} \tag{4.7}$$

The entries of $\frac{dL}{d\mathbf{b}}$ can be computed directly from the definition of the loss, as summarized in Table 4.1. For $\frac{\partial^2 F}{\partial \mathbf{b}\partial\theta}$, notice first that, from the definition of F (Eq. 4.1),

$$\frac{dF}{d\mathbf{b}} = \mathbf{w}(\mathbf{x}) \odot (\log \mathbf{b} + 1) + \mathbf{v}(\mathbf{x}), \tag{4.8}$$

and so,

$$\frac{d^2 F}{d\mathbf{b}d\theta} = (\log \mathbf{b} + 1) \odot \frac{d\mathbf{w}(\mathbf{x})}{d\theta} + \frac{d\mathbf{v}(\mathbf{x})}{d\theta}.$$

The derivatives of $\mathbf{w}(\mathbf{x})$ and $\mathbf{v}(\mathbf{x})$ with respect to θ depend on the parametrization of the model. (In general, there is no weight sharing between $\mathbf{w}$ and $\mathbf{v}$, and so one of these derivatives will be zero.) Notice in particular that it is possible to fit not only the factors (v), but also the entropy terms (w). These will be discussed in more detail in Section 6.1.

For this objective function,

$$D = \frac{\partial^2 F}{\partial \mathbf{b}\partial \mathbf{b}^T} = \mathrm{diag}(\mathbf{w}(\mathbf{x}) \oslash \mathbf{b}). \tag{4.9}$$

Finally, notice that in Eq. 4.7, $\mathbf{m}^T = \frac{dL}{d\mathbf{b}^T}\big(D^{-1}A^T(AD^{-1}A^T)^{-1}AD^{-1} - D^{-1}\big)$ does not depend on which parameter is being differentiated. Hence, this can be computed once, and reused to compute each parameter derivative by $\frac{dL}{d\theta} = \mathbf{m}^T \frac{\partial^2 F}{\partial \mathbf{b}\partial\theta}$. This can give a large computational savings when there are many parameters, as solving the linear system required to find $\mathbf{m}^T$ can dominate in large models.

For concreteness, the full procedure to calculate $dL/d\theta$ summarized as Alg. 3.

Algorithm 3 How to calculate $dL/d\theta$.

1. Input some training element $(\hat{\mathbf{y}}, \hat{\mathbf{x}})$, and parameters $\boldsymbol{\theta}$.
2. Perform the optimization (Section 4.3):

$$\begin{array}{rrl} \text{minimize} & F(\mathbf{b}) & = \mathbf{w}(\hat{\mathbf{x}})^T(\mathbf{b} \odot \log \mathbf{b}) + \mathbf{v}(\hat{\mathbf{x}})^T\mathbf{b} \\ \text{s.t.} & A\mathbf{b} & = \mathbf{d} \\ & \mathbf{b} & \geq \mathbf{0}. \end{array}$$

3. Calculate $\frac{dL}{d\mathbf{b}}$. (Table 4.1)
4. $D^{-1} \leftarrow \text{diag}(\mathbf{b} \oslash \mathbf{w}(\mathbf{x}))$
5. Solve a linear system to obtain $\mathbf{m}^T \leftarrow \frac{dL}{d\mathbf{b}^T}\big(D^{-1}A^T(AD^{-1}A^T)^{-1}AD^{-1} - D^{-1}\big)$.
6. Calculate the gradient. $\forall j$, $\frac{dL}{d\theta_j} \leftarrow \mathbf{m}^T \frac{\partial^2 F}{\partial \mathbf{b} \partial \theta_j}$.

This can be employed in any standard gradient-based optimization algorithm, e.g. L-BFGS, stochastic gradient descent, etc.

4.7 Discussion

This chapter considers a CRF as simply defining an objective function that, when optimized, gives predicted marginals. This *mapping* from input to predicted marginals is fit in learning. This strategy has the advantage that learning compensates for defects in inference. However, there are two major downsides.

The first issue is efficiency. Solving the optimization problem takes a varying number of iterations, each of which has a significant cost.

The second issue is the limitation to convex optimizations. (Note that this could be relaxed somewhat by only enforcing that F is convex over the constraint set[19]). In particular, the Bethe approximation is non-convex, and it has been suggested[20] that this leads to better predicted marginals when optimized successfully. The next chapter presents a different strategy that can cope with non-convex entropies.

Univariate Losses	L	$\frac{dL}{db(y_i)}$
Cond. Likelihood	$\sum_i -\log b(\hat{y}_i)$	$[y_i = \hat{y}_i]\frac{-1}{b(y_i)}$
Quadratic	$\sum_i \big(-2b(\hat{y}_i) + \sum_{y_i} b(y_i)^2\big)$	$-2[y_i = \hat{y}_i] + 2b(y_i)$
Smooth Class.	$\sum_i \sigma\big(\lambda(\max_{y_i \neq \hat{y}_i} b(y_i) - b(\hat{y}_i))\big)$	$\sigma' \cdot \lambda([y_i = \arg\max_{y_i \neq \hat{y}_i} b(y_i)] - [y_i = \hat{y}_i]\big)$
Clique Losses	L	$\frac{dL}{db(\mathbf{y}_c)}$
Cond. Likelihood	$\sum_c -\log b(\hat{\mathbf{y}}_c)$	$[\mathbf{y}_c = \hat{\mathbf{y}}_c]\frac{-1}{b(\mathbf{y}_c)}$
Quadratic	$\sum_c \big(-2b(\hat{\mathbf{y}}_c) + \sum_{\mathbf{y}_c} b(\mathbf{y}_c)^2\big)$	$-2[\mathbf{y}_c = \hat{\mathbf{y}}_c] + 2b(\mathbf{y}_c)$
Smooth Class.	$\sum_c \sigma\big(\lambda(\max_{\mathbf{y}_c \neq \hat{\mathbf{y}}_c} b(\mathbf{y}_c) - b(\hat{\mathbf{y}}_c))\big)$	$\sigma' \cdot \lambda([\mathbf{y}_c = \arg\max_{\mathbf{y}_c \neq \hat{\mathbf{y}}_c} b(\mathbf{y}_c)] - [\mathbf{y}_c = \hat{\mathbf{y}}_c]\big)$

Table 4.1: Loss functions and derivatives

4.8 Appendix

Here, the result is proven for a vector of parameters $\boldsymbol{\theta}$.

Claim: Let $F(\mathbf{b}, \theta)$ be strictly convex over $\mathbf{b}$. If $\mathbf{b}(\boldsymbol{\theta}) = \arg\min_{\mathbf{b}} F(\mathbf{b}, \boldsymbol{\theta})$ such that $A\mathbf{b} = \mathbf{d}$, then

$$\frac{\partial \mathbf{b}}{\partial \boldsymbol{\theta}^T} = \big(D^{-1}A^T(AD^{-1}A^T)^{-1}AD^{-1} - D^{-1}\big)\frac{\partial^2 F}{\partial \mathbf{b} \partial \boldsymbol{\theta}^T},$$

where $D := \frac{\partial^2 F}{\partial \mathbf{b} \partial \mathbf{b}^T}$.

Proof:

First, create a Lagrangian, enforcing the constraint.

$$\mathcal{L} = F(\mathbf{b}, \boldsymbol{\theta}) + \boldsymbol{\lambda}^T(A\mathbf{b} - \mathbf{d}).$$

The basic idea is to consider the response of the joint vector $[\mathbf{b}, \boldsymbol{\lambda}]$ to a change in $\boldsymbol{\theta}$. The implicit functional relationship is

$$\begin{bmatrix} \frac{\partial \mathcal{L}(\mathbf{b},\boldsymbol{\lambda},\boldsymbol{\theta})}{\partial \mathbf{b}} \\ \\ \frac{\partial \mathcal{L}(\mathbf{b},\boldsymbol{\lambda},\boldsymbol{\theta})}{\partial \boldsymbol{\lambda}} \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ \\ \mathbf{0} \end{bmatrix}.$$

So the derivatives must, by Eq. 4.5, satisfy the linear system

$$\begin{bmatrix} \frac{\partial \mathbf{b}}{\partial \boldsymbol{\theta}^T} \\ \\ \frac{\partial \boldsymbol{\lambda}}{\partial \boldsymbol{\theta}^T} \end{bmatrix} = - \begin{bmatrix} \frac{\partial^2 \mathcal{L}}{\partial \mathbf{b} \partial \mathbf{b}^T} & \frac{\partial^2 \mathcal{L}}{\partial \mathbf{b} \partial \boldsymbol{\lambda}^T} \\ \\ \frac{\partial^2 \mathcal{L}}{\partial \boldsymbol{\lambda} \partial \mathbf{b}^T} & \frac{\partial^2 \mathcal{L}}{\partial \boldsymbol{\lambda} \partial \boldsymbol{\lambda}^T} \end{bmatrix}^{-1} \begin{bmatrix} \frac{\partial^2 \mathcal{L}}{\partial \mathbf{b} \partial \boldsymbol{\theta}^T} \\ \\ \frac{\partial^2 \mathcal{L}}{\partial \boldsymbol{\lambda} \partial \boldsymbol{\theta}^T} \end{bmatrix}.$$

Making the substitutions

$$\begin{aligned} \frac{\partial^2 \mathcal{L}}{\partial \mathbf{b} \partial \boldsymbol{\lambda}^T} &= A \\ \frac{\partial^2 \mathcal{L}}{\partial \boldsymbol{\lambda} \partial \mathbf{b}^T} &= A^T \\ \frac{\partial^2 \mathcal{L}}{\partial \mathbf{b} \partial \boldsymbol{\theta}^T} &= \frac{\partial^2 F}{\partial \mathbf{b} \partial \boldsymbol{\theta}^T} \\ \frac{\partial^2 \mathcal{L}}{\partial \boldsymbol{\lambda} \partial \boldsymbol{\lambda}^T} &= 0 \\ \frac{\partial^2 \mathcal{L}}{\partial \boldsymbol{\lambda} \partial \boldsymbol{\theta}^T} &= 0, \end{aligned}$$

and defining $D = \frac{\partial^2 \mathcal{L}}{\partial \mathbf{b} \partial \mathbf{b}^T} = \frac{\partial^2 F}{\partial \mathbf{b} \partial \mathbf{b}^T}$, this is equivalent to the system

$$\begin{bmatrix} \frac{\partial \mathbf{b}}{\partial \boldsymbol{\theta}^T} \\ \\ \frac{\partial \boldsymbol{\lambda}}{\partial \boldsymbol{\theta}^T} \end{bmatrix} = - \begin{bmatrix} D & A^T \\ \\ A & 0 \end{bmatrix}^{-1} \begin{bmatrix} \frac{\partial^2 F}{\partial \mathbf{b} \partial \boldsymbol{\theta}^T} \\ \\ 0 \end{bmatrix}.$$

One can recover $\frac{\partial \mathbf{b}}{\partial \boldsymbol{\theta}^T}$ directly by solving this system. Alternatively, consider the inverse of the central matrix. If its entries are X, Y, Z, U, by definition it must satisfy

$$\begin{bmatrix} D & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} X & Y \\ Z & U \end{bmatrix} = \begin{bmatrix} I & 0 \\ 0 & I \end{bmatrix}. \tag{4.10}$$

Two identities that follow directly from Eq. 4.10 are

$$\begin{aligned} DX + A^T Y &= I, \\ AX &= 0. \end{aligned}$$

Solving these for X gives

$$X = D^{-1} - D^{-1}A^T(AD^{-1}A^T)^{-1}AD^{-1}.$$

Finally, observing that $\frac{\partial \mathbf{b}}{\partial \boldsymbol{\theta}^T} = -X\frac{\partial^2 F}{\partial \mathbf{b}\partial \boldsymbol{\theta}^T}$ gives the result.

□

Chapter 5

Procedural Fitting

5.1 Message-passing algorithms as mappings.

The previous chapter treated graphical models as simply defining an energy function. Given an observation $\mathbf{x}$, this function is minimized to give predicted marginals. Learning treats this function as essentially a black-box mapping: we want to fit parameters so that the predicted marginals are good (as quantified by a loss function).

There are two major downsides to the previous strategy: efficiency and the restriction to convex energy functions. Performing a nonlinear optimization like in Algorithm 1 is possible, even in large-scale situations. However, often simply running a few iterations of a message passing algorithm like Loopy Belief-Propagation appears to give good results in far less time.

It has been suggested that non-convex entropy approximations like the Bethe approximation can yield better marginals if minimized successfully. But the lack of confidence in reaching the global minimum complicates learning, and makes it difficult to speak of a "mapping" from an observation to marginals.

This chapter treats graphical models as defining a different type of mapping. Rather than thinking of a model producing an *energy function*, think of it producing

a *message passing algorithm*. If one uses a fixed update order, and a fixed number of iterations, a message passing algorithm can be considered a deterministic mapping. This is true even if the algorithm does not reach the global minima, or even fails to converge: parameters will be fit so that approximate marginals after the fixed number of iterations are optimal.

5.2 Automatic Differentiation

Automatic differentiation is a technique to compute gradients of functions. It is distinct from both numerical (or finite difference) differentiation and symbolic differentiation. In particular, we are interested here in "reverse mode" automatic differentiation (RAD).

Suppose one has implemented an algorithm that takes n inputs $x_1, ..., x_n$, and produces a single output. RAD will transform this algorithm into one that computes the gradient of that function with respect to $x_1, ..., x_n$. In machine learning applications, one will typically have a function taking parameters of a model as input and outputting a loss function, measuring how well those parameters fit some training data. One would like the gradient so as to optimize that loss. Aside from the convenience of not needing to derive gradients, RAD is very computationally efficient: The resulting gradient algorithm has *the same* computational complexity as the original function.

The basic idea of RAD is to transform the original algorithm into an expression graph, or a series of assignments to the results of basic operations.

Forward Propagation

1. For $i = n + 1, n + 2, \ldots N$:

   (a) $x_i \leftarrow f_i(\mathbf{x}_{\pi(i)})$

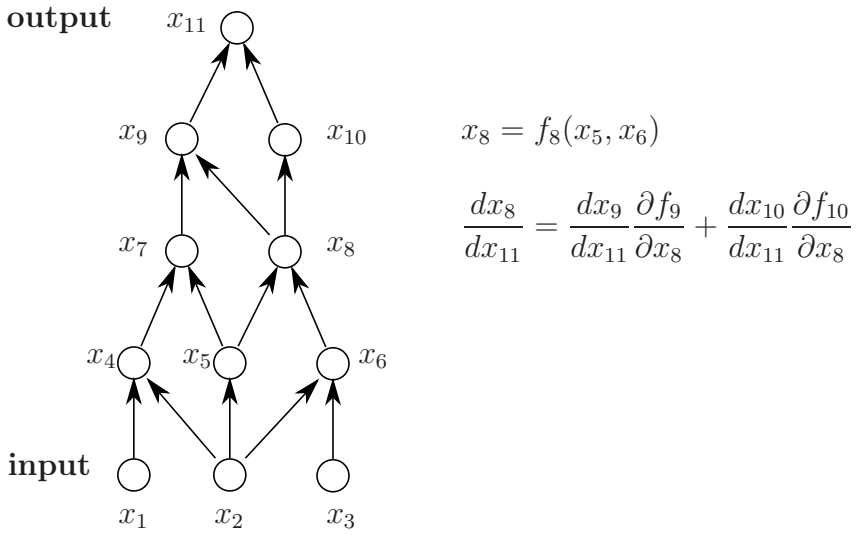


Figure 5.1: A simple expression graph with $n = 3$ inputs, and a total of $N = 11$ variables.

Here, each f_i is some basic operation, for example binary addition, multiplication, or some unary operation such as a logarithm. It can be essentially arbitrary, but it must be differentiable.

Now, we would like to compute dx_N/dx_i for all i. We see from the chain rule that this can be computed from the derivatives of all immediate children of i by

$$\frac{dx_N}{dx_i} = \sum_{j:i\in\pi(j)} \frac{dx_N}{dx_j}\frac{\partial x_j}{\partial x_i}.$$

Hence, the derivatives can be computed by the following algorithm.

Back-Propagation

1. $\frac{dx_N}{dx_N} \leftarrow 1$
2. For $i = N-1, N-2, \ldots 1$:
   (a) $\frac{dx_N}{dx_i} \leftarrow \sum_{j:i\in\pi(j)} \frac{dx_N}{dx_j}\frac{\partial f_j}{\partial x_i}$

This is illustrated on a very simple expression graph in Fig. 5.1. For more details, see standard references on automatic differentiation [21].

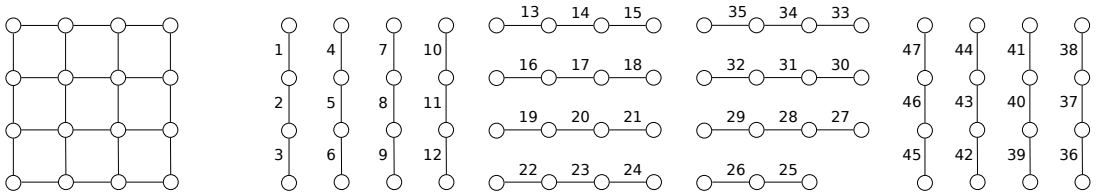


Figure 5.2: One iteration of updates for a a "grid" graph.

5.3 Procedurally fit CRFs

To procedural fit a CRF, one simply writes a routine to perform a fixed sequence of message passing updates, then compute the loss corresponding to the predicted marginals. Applying automatic differentiation to this routine produces the gradient of the loss with respect to the parameters of the model, which can be employed in a gradient-based optimization algorithm.

The experiments below will use "grid" models, like Fig. 5.2. Updates are ordered by cliques. First, each vertical clique is updated, each line in order, starting in the upper-left corner. Next, horizontal connections are updated, also starting from the upper-left corner. Finally, the process is repeated in the reverse order. (i.e. horizontal connections are updated from the lower-right corner followed by vertical connections also from the lower-right.) This is illustrated in Figure 5.2 for a 4x4 grid. This order is chosen so that information on any part of the graph can reach every other part in one iteration.

To update a clique, first all messages into that clique from connected variables are updated. These messages are then used to update the messages out of the clique. Thus, in the notation of Section 2.2.2, when updating clique c, one first updates $m_{i \to c}$, for all $i \in c$ and then updates $m_{c \to i}$.

In the experiments below, all messages are initialized to one.

5.4 Discussion

Though this strategy was applied here to loopy belief propagation, it could be applied to different message passing algorithms as well, such as Mean Field (Section 2.2.5) Tree-reweighted belief propagation[22], or expectation propagation[23].

Another way to potentially improve performance would be to increase the flexibility of the message updates. One way to do this, which has worked well in preliminary experiments, is to give each iteration of updates its own set of parameters. (Usually, one would do this by first fitting a model with fixed parameters for initialization.) Done this way, the different updates can provide a kind of automatic convergence control for the beliefs.

Chapter 6

Experiments

6.1 Parametrization

This section describes how $\mathbf{w}(\mathbf{x})$ and $\mathbf{v}(\mathbf{x})$ can be parametrized in the case of linear features. This allows a unified treatment of different types of input variables $\mathbf{x}$ (discrete or continuous, etc.) Thus, abstractly, the weighting functions are given by

$$v(\mathbf{y}_c, \mathbf{x}) = \boldsymbol{\theta}_v(\mathbf{y}_c)^T \mathbf{f}_c(\mathbf{x})$$
$$v(y_i, \mathbf{x}) = \boldsymbol{\theta}_v(y_i)^T \mathbf{f}_i(\mathbf{x}),$$

where $\mathbf{f}_c(\mathbf{x})$ and $\mathbf{f}_i(\mathbf{x})$ are features of the input[1]. Here, the notation $\boldsymbol{\theta}_v(\mathbf{y}_c)$ is meant to indicate that each clique configuration has its own set of parameters. These can be specified arbitrarily by the "user" of the algorithm, though several typical cases are given below. Now, given these definitions of features, we trivially have

[1]Note that these features could also be expressed as, e.g., $v(\mathbf{y}_c, \mathbf{x}) = \boldsymbol{\theta}^T \mathbf{h}_c(\mathbf{y}_c, \mathbf{x})$, where the feature vector $\mathbf{h}_c$ is now the cross product of the input features $\mathbf{f}_c(\mathbf{x})$ and indicator functions on the values $\mathbf{y}_c$. Thus, the CRF resulting from these functions is a member of the exponential family.

$$\frac{dv(\mathbf{y}_c, \mathbf{x})}{d\boldsymbol{\theta}_v(\mathbf{y}_c)} = \mathbf{f}_c(\mathbf{x})$$
$$\frac{dv(y_i, \mathbf{x})}{d\boldsymbol{\theta}_v(y_i)} = \mathbf{f}_i(\mathbf{x}).$$

This framework can be used with or with out weight sharing, and for discrete or continuous input. The weights w are defined similarly, in terms of some features $\mathbf{g}_c$ and $\mathbf{g}_c$.

$$w(\mathbf{y}_c, \mathbf{x}) = \boldsymbol{\theta}_w(\mathbf{y}_c)^T \mathbf{g}_c(\mathbf{x})$$
$$w(y_i, \mathbf{x}) = \boldsymbol{\theta}_w(y_i)^T \mathbf{g}_i(\mathbf{x})$$

6.1.1 Relationship to traditional CRFs

Theoretically, the features $\mathbf{f}_c, \mathbf{f}_i, \mathbf{g}_c, \mathbf{g}_i$ can depend on $\mathbf{x}$, in arbitrary, "user-specified" ways. However, for concreteness, typical cases most similar to traditional CRF approaches are explored here. A common CRF for images, and the one that will be used for experiments below is

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_c \psi(\mathbf{y}_c) \prod_i \psi(y_i, x_i).$$

Thus, there are some factors $\psi(\mathbf{y}_c)$ defined on neighboring pairs c, not dependent on the input, and univariate factors $\psi(y_i, x_i)$ depending only on the input at a single location. For this case, the "features" for cliques would simply be constant, the "work" being entirely done by $\boldsymbol{\theta}_v(\mathbf{y}_c)$.

$$f_c(\mathbf{x}) = 1, \quad \boldsymbol{\theta}_v(\mathbf{y}_c) \text{ arbitrary}$$

Now if the input is real-valued (e.g. a grayscale image), reasonable univariate features would be a constant, plus the value at that node.

$$f_i(\mathbf{x}) = \begin{bmatrix} 1 \\ \\ x_i \end{bmatrix}, \quad \boldsymbol{\theta}_v(y_i) \text{ arbitrary}$$

If, on the other hand, the input is discrete valued, features might be indicator functions for each possible value of x_i.

$$f_i(\mathbf{x}) = \begin{bmatrix} [x_i = 1] \\ \\ [x_i = 2] \\ \\ \vdots \\ \\ [x_i = D] \end{bmatrix}, \quad \boldsymbol{\theta}_v(y_i) \text{ arbitrary}$$

Taking the inner product of these indicator functions with an arbitrary $\mathbf{a}(y_i)$ is equivalent to simply defining $w(y_i, x_i)$ by a "table", but removes the need to treat discrete input as a special case.

Traditionally, the entropy terms w are not functions of $\mathbf{x}$ at all. When taking the Bethe approximation, one would fix

$$g_c(\mathbf{x}) = 1, \quad \boldsymbol{\theta}_w(\mathbf{y}_c) = 1,$$
$$g_i(\mathbf{x}) = 1, \quad \boldsymbol{\theta}_w(y_i) = n_i,$$

and not modify $\boldsymbol{\theta}_w$ during learning. Alternatively, one could "fit the entropy approximation" during learning. This would mean taking values for each variable or clique independent of $\mathbf{y}$.

$$g_c(\mathbf{x}) = 1, \quad \boldsymbol{\theta}_w(\mathbf{y}_c) = \theta_{wc},$$
$$g_i(\mathbf{x}) = 1, \quad \boldsymbol{\theta}_w(y_i) = \theta_{wi},$$

Yet more general would be, just as for v above, to take $\mathbf{g}_c, \mathbf{g}_i$ as rich features of the input and allow $\boldsymbol{\theta}_w(\mathbf{y}_c)$ and $\boldsymbol{\theta}_w(y_i)$ to be arbitrary.

6.2 Binary Digits

The binary digit dataset consists of binarized 28x28 handwritten digits $\hat{\mathbf{y}}$ from the MNIST database. These are corrupted with various amounts to form the input $\hat{\mathbf{x}}$. There is a training set of 90 images (10 of each digit 1-9), and a similar set of 90 test images. The input was corrupted with various amounts of noise: 10%, 30%, 50% and 70%. (The amount of noise indicates the fraction of pixels that are set randomly.) The various loss functions used for training are shown in Table 6.1.

Because `pseudo` and `convex` are the only convex loss functions here, convex is first fit, and then used to initialize the parameters of all other loss functions. Training uses L-BFGS for the learning optimization. The primal optimization algorithm from Section 4.4 was used for optimizing the beliefs of `convex` and the implicit losses. These also all use the same simple convex entropy approximation, namely $w(\mathbf{y}_c) = 1, w(y_i) = .01$. Loopy belief propagation is used for the test stage for the model trained with `pseudo`.

Figures 6.1-6.4 show the errors for all the different methods, evaluated in various ways. Figures 6.5-6.8 show example results. `pseudo` and `lbp` based learning perform acceptably for low noise levels, but are disastrous above 50% noise. The `convex` approach is robust to high amounts of noise, but does not perform quite as well as the proposed methods. For example, with 50% noise, `convex` has a univariate classifica-

Abbreviation	Loss
pseudo	Conditional pseudolikelihood
lbp	Loopy belief propagation approximation of conditional likelihood.
convex	Convex entropy approximation of conditional likelihood.
ucl	Univariate conditional likelihood (implicit fitting)
ccl	Clique conditional likelihood (implicit fitting)
uquad	Univariate quadratic (implicit fitting)
cquad	Clique quadratic (implicit fitting)
ucl-k	Univariate conditional likelihood with k iterations (procedural fitting)
ccl-k	Clique conditional likelihood with k iterations (procedural fitting)
uquad-k	Univariate quadratic with k iterations (procedural fitting)
cquad-k	Clique quadratic with k iterations (procedural fitting)

Table 6.1: Loss function abbreviations.

tion error of .0716, while `uquad` (implicit fitting) has .0700, and `uquad-4` (procedural fitting with 4 iterations) has .0601. Again, at 70% noise, `uquad` (procedural fitting) slightly edges out `convex` (.0134 vs. .0158), while `uquad-4` (procedural fitting) at .0106 again beats both.

The results of these experiments are roughly as follows:

- With a given method (implicit fitting or procedural fitting) which specific loss function is used makes relatively little difference.
- Implicit fitting works similarly to the convex likelihood, with somewhat better results on the higher noise levels, depending on the specific loss functions. The results are also visually very similar. (This is not so surprising, given that these methods use the same entropy approximation and belief optimization algorithm.)
- Procedural fitting generally performs better than implicit fitting.

- The results of procedural fitting depend surprisingly little on the number of iterations used. In general, four iterations do better than one, but this effect is neither strong nor universal.

6.3 Fitting Entropy Approximations

Implicit fitting has one advantage not taken advantage of in the above experiments. Namely, it is possible to also fit entropy approximations, so as to give the best possible predictions. Here, the binary digits dataset with 50% noise is used to test the effects of fitting entropy terms like this. The univariate conditional likelihood was fit with eight different entropy approximations:

- `A`: $w(\mathbf{y}_c) = 1, w(y_i) = .01$
- `B`: $w(\mathbf{y}_c) = 1, w(y_i) = .1$
- `C`: $w(\mathbf{y}_c) = 1, w(y_i) = 1$
- `r1-r5`: Each entry of $w(\mathbf{y}_c)$ and $w(y_i)$ chosen randomly from $[0, 1]$.

After fitting with the entropy fixed, another learning optimization was run to adjust the entropy parameters. The results are shown in Figures 6.9 and 6.10. The following conclusions are available from the results.

- Entropies `A`, `B`, and `C` are progressively less close to the Bethe approximation. `A`, the closest approximation, does give slightly better results.
- One particular randomly chosen entropy (`r3`) happens to work better than the others, including the more traditional approximations (`A`,`B`,`C`).
- Fitting the entropies improves results considerably, and gives very similar results regardless of the initial entropy. The final univariate classifications errors (.050-.059) are slightly better than the results for procedural fitting (.059-.063).

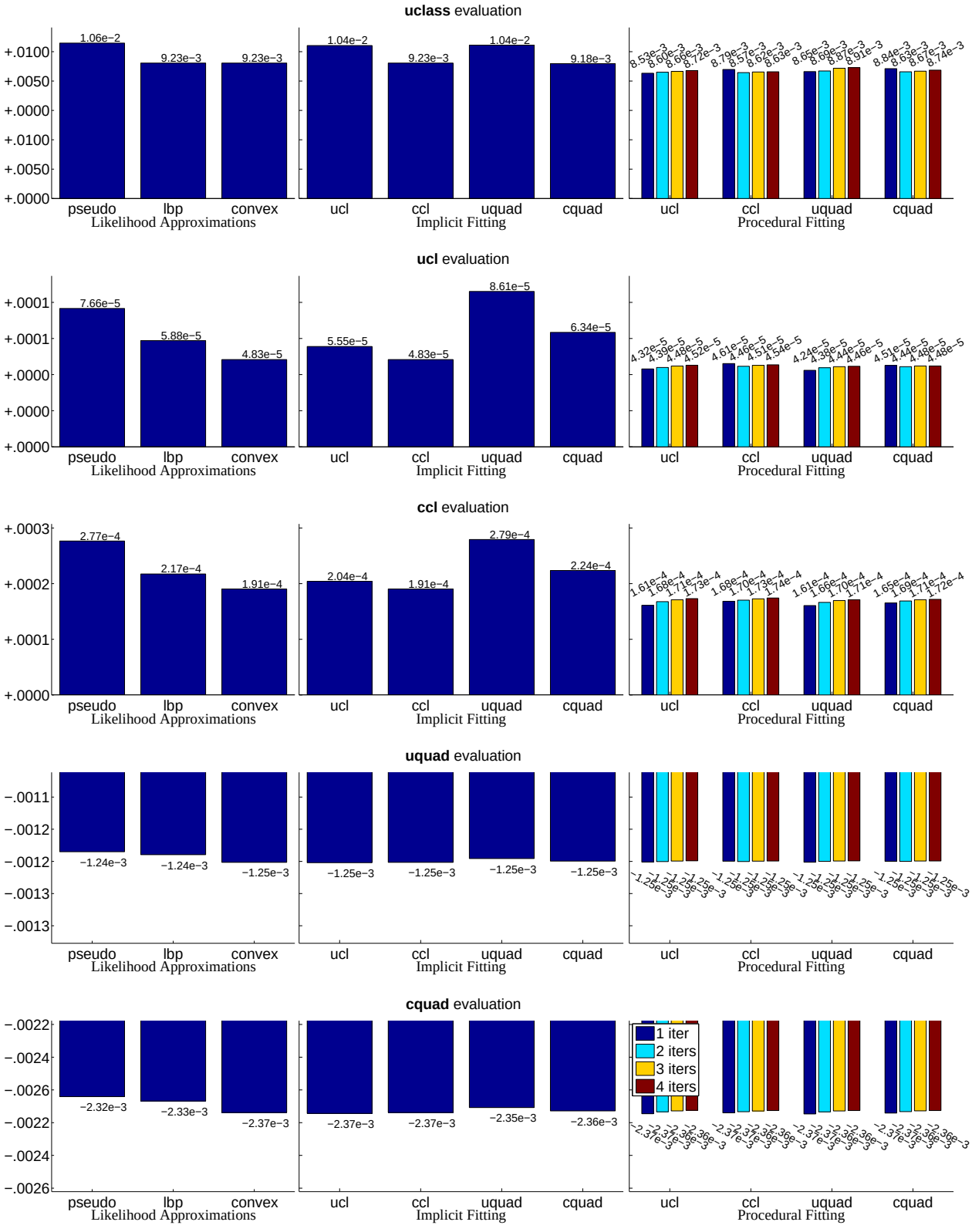


Figure 6.1: Binary Digit Errors-10% noise

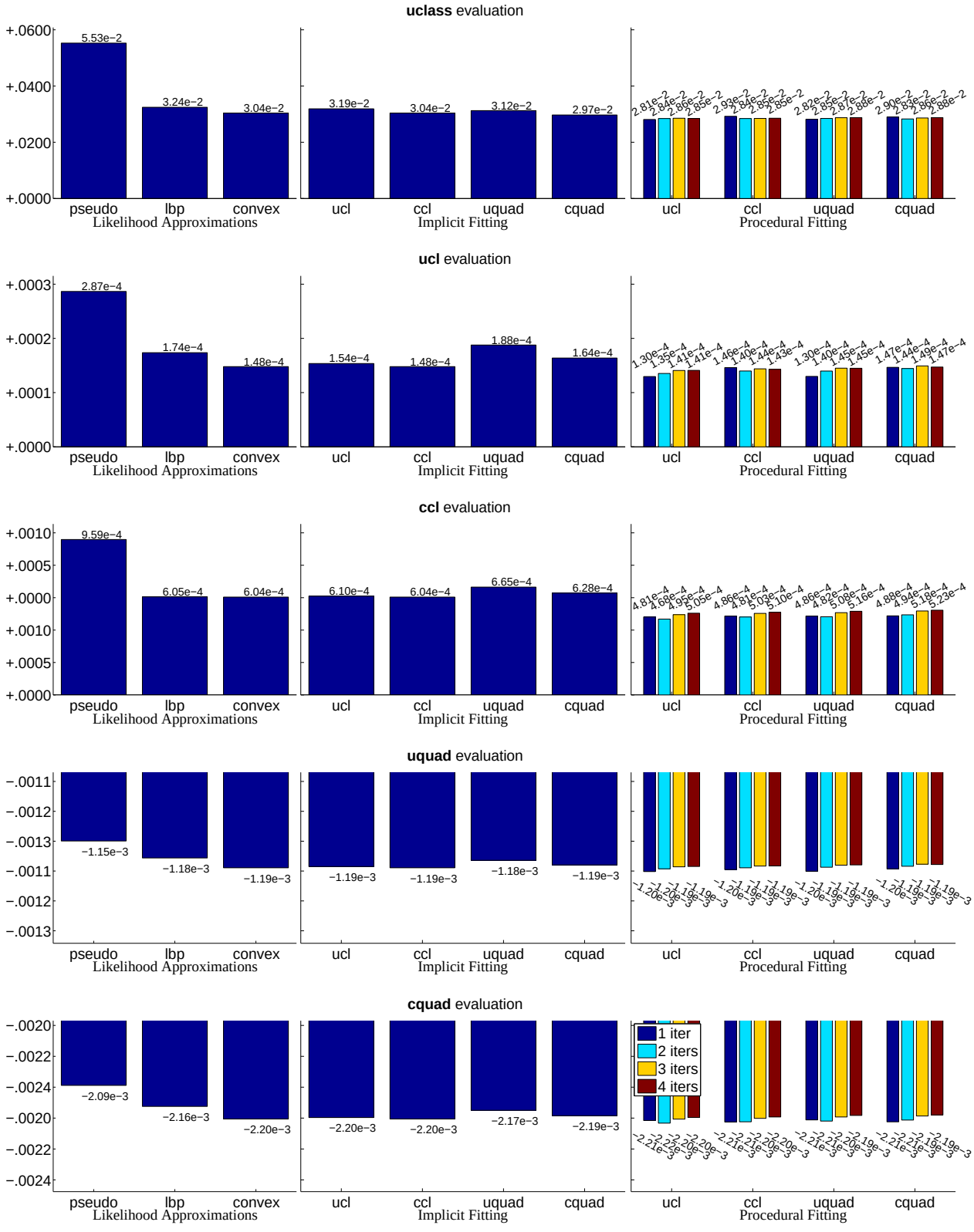


Figure 6.2: Binary Digit Errors-30% noise

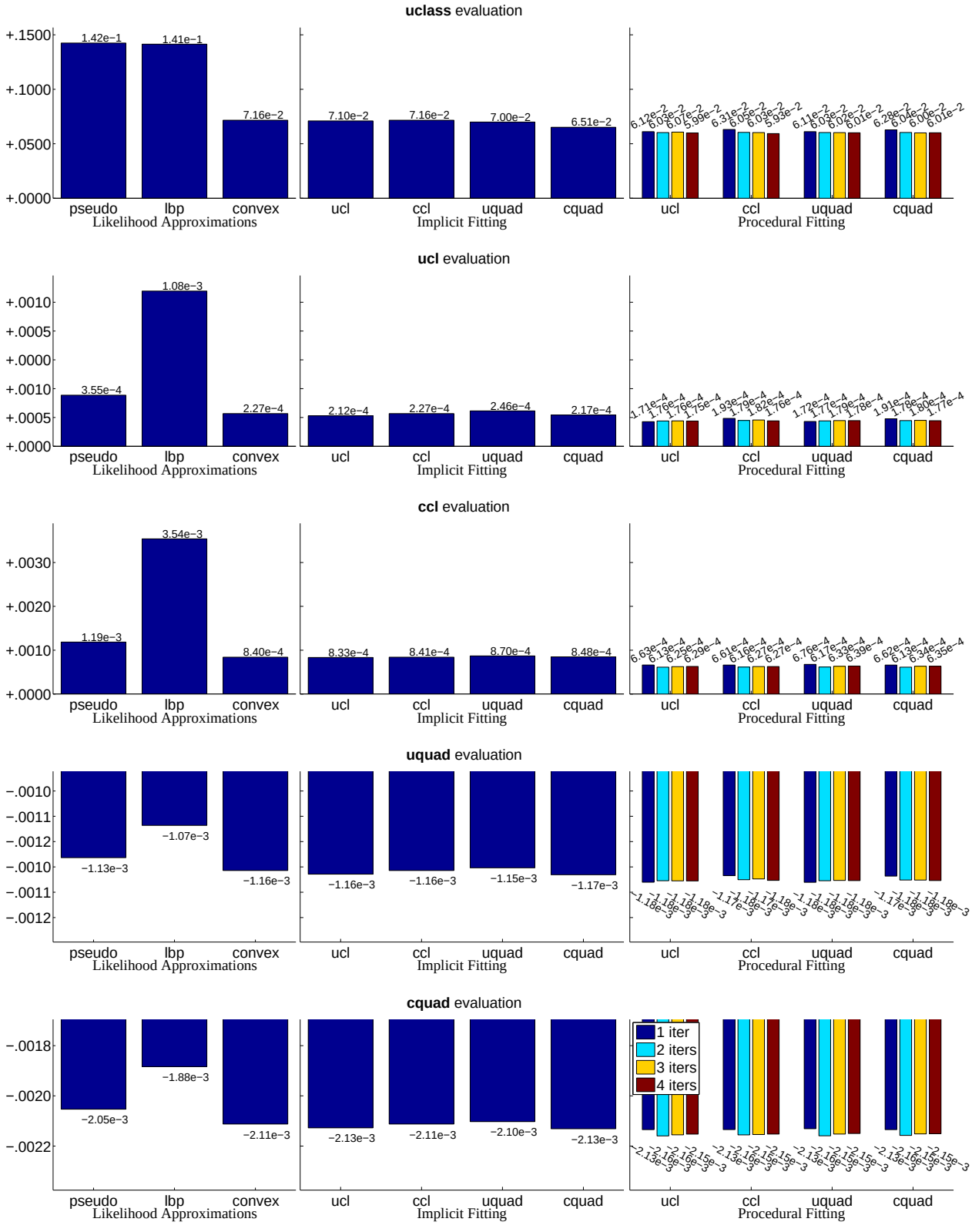


Figure 6.3: Binary Digit Errors-50% noise

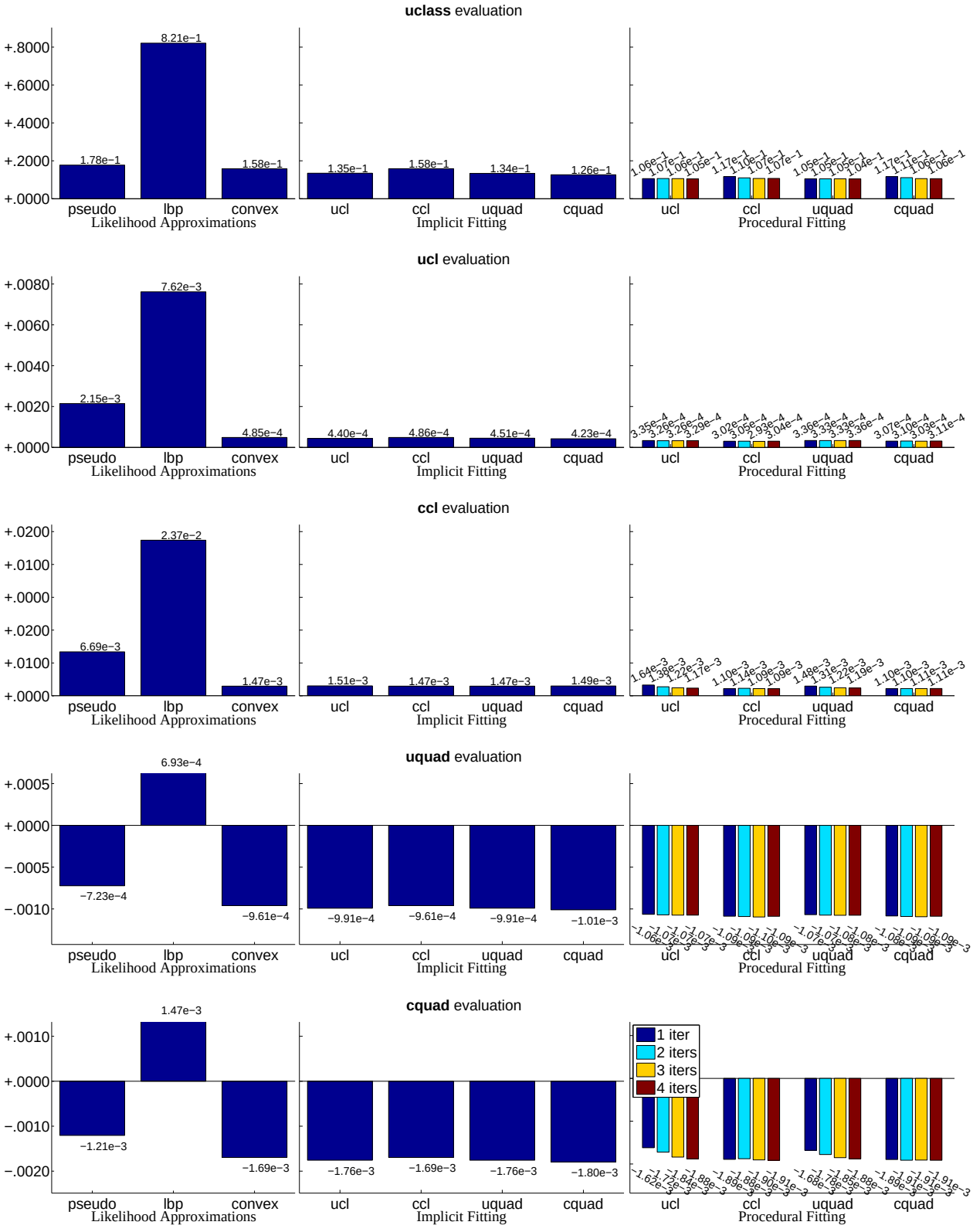


Figure 6.4: Binary Digit Errors-70% noise

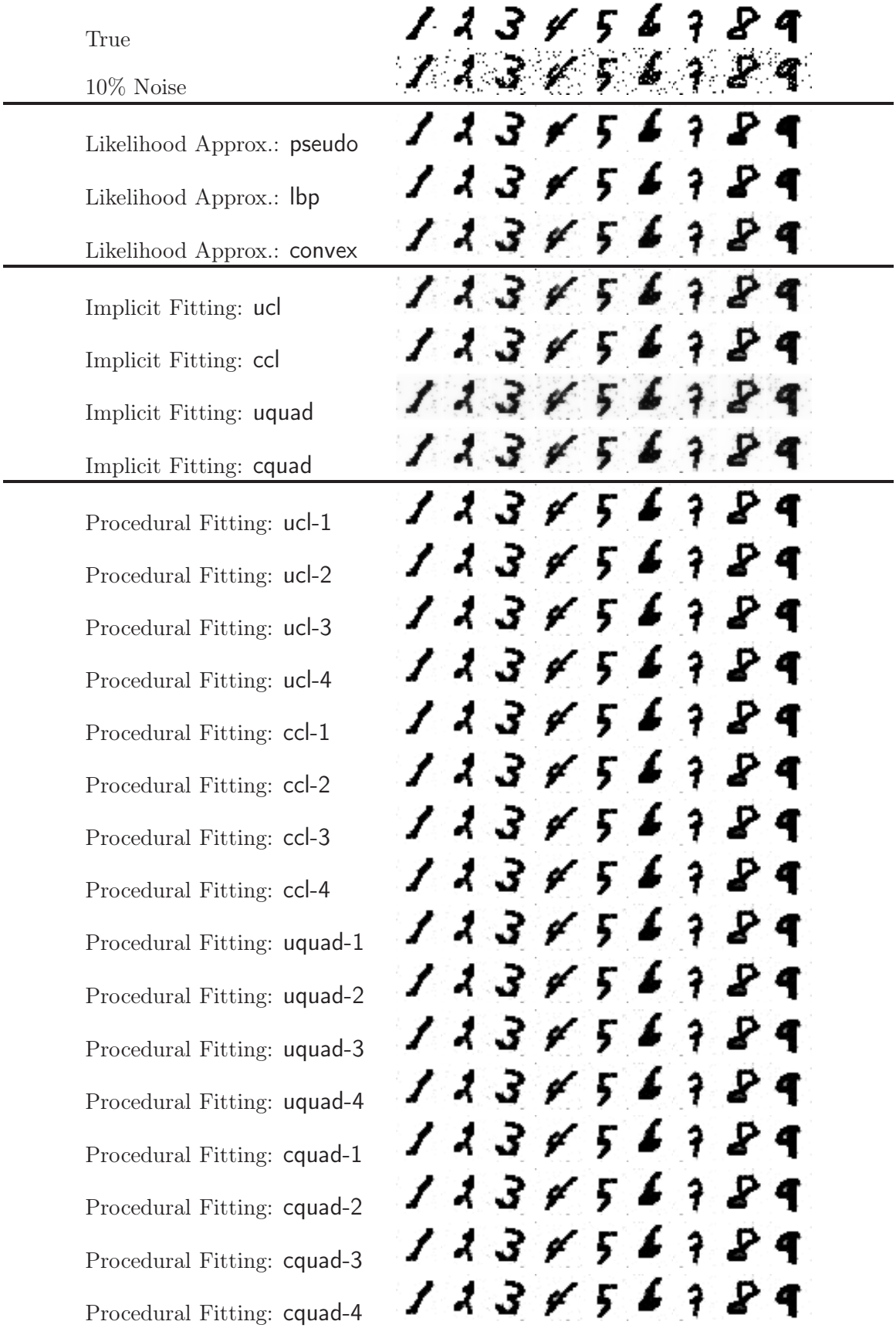


Figure 6.5: Example Binary Digit Results- 10% noise

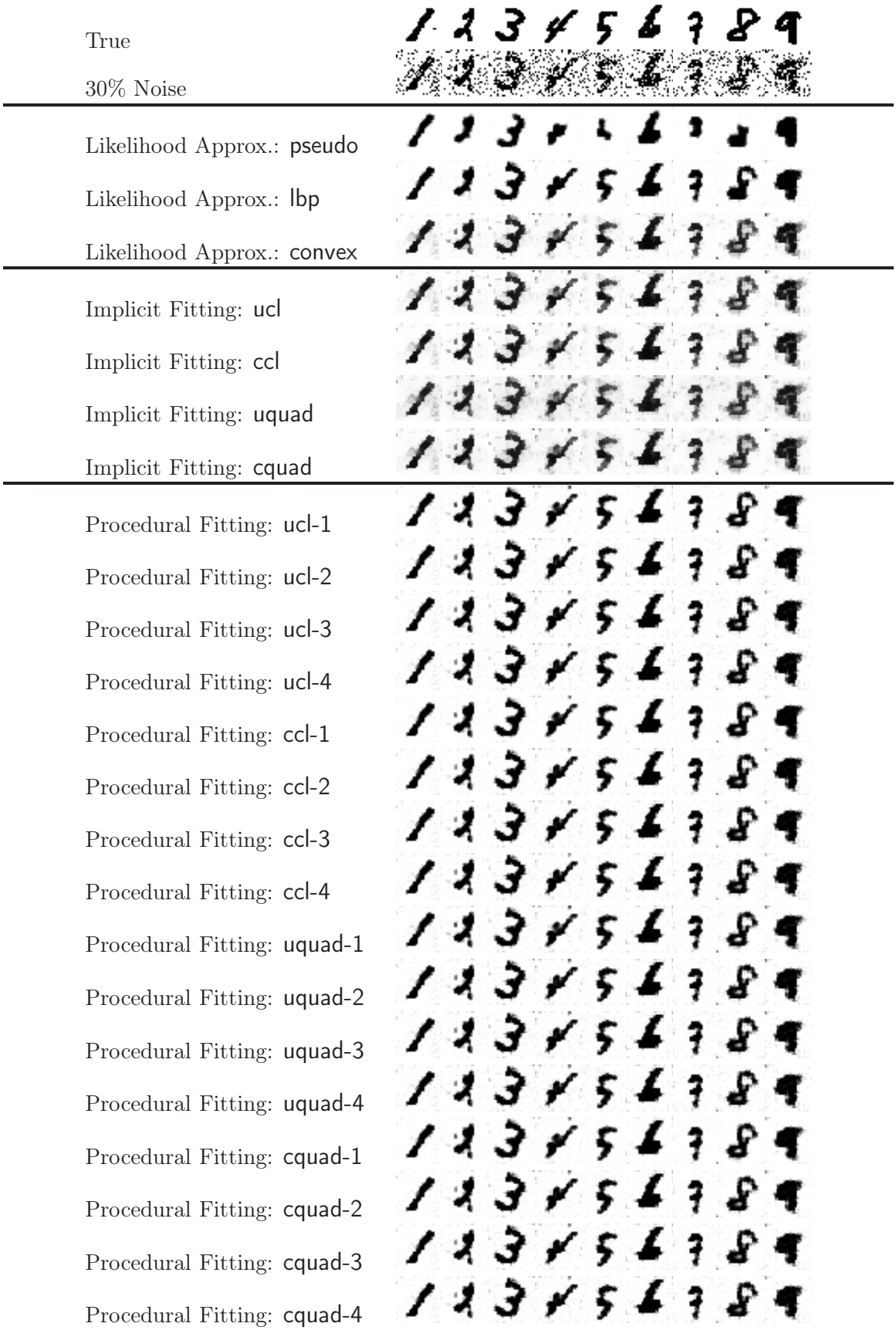


Figure 6.6: Example Binary Digit Results- 30% noise

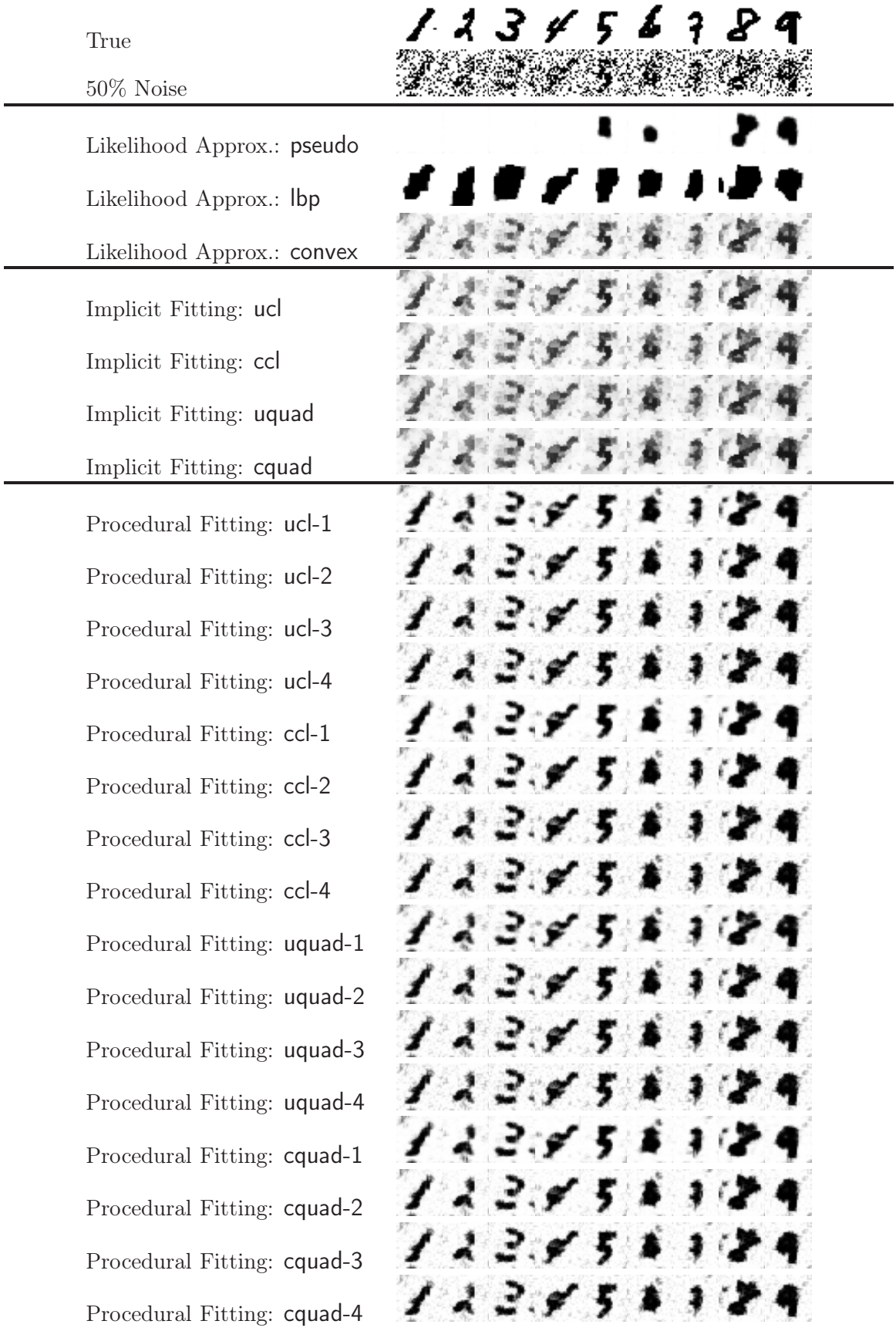


Figure 6.7: Example Binary Digit Results- 50% noise

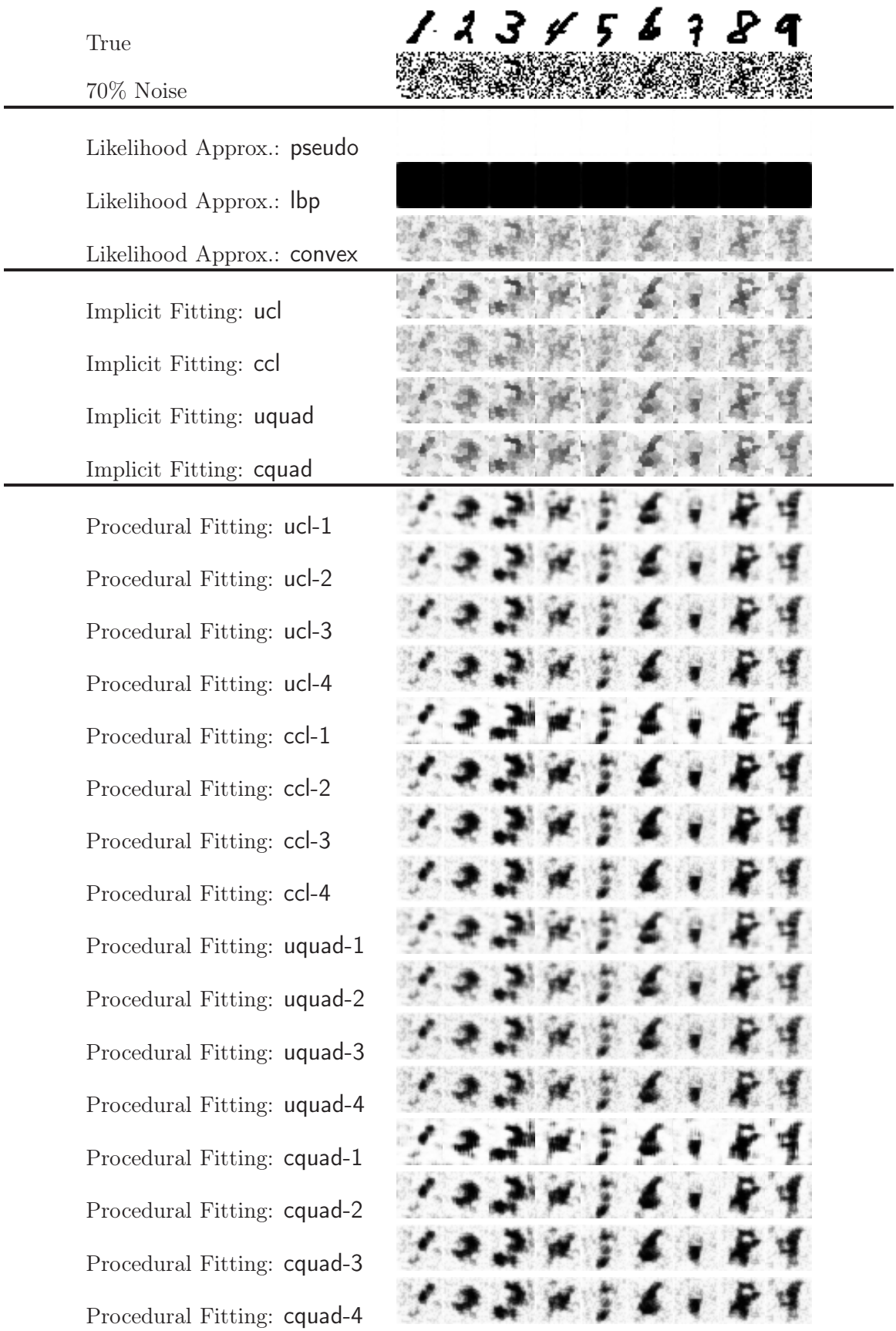


Figure 6.8: Example Binary Digit Results- 70% noise

It should be said that fitting entropy terms like this is a somewhat strange thing to do. Though no theory suggests doing this, neither does any theory suggest it is a *bad* idea. Intuitively speaking, allowing entropy terms to vary simply gives the learning algorithm "more knobs to twiddle", and so it is not so surprising that this improves the results.

The results here allow one entropy value for each clique or univariate configuration, independently of the input. One could go further, and allow the entropy terms to use full features of the input, as described in Section 6.1.

6.4 Varying the number of iterations of Procedural Fitting

When procedurally fit models are trained with a fixed number of iterations, how fragile are they to a change in that number? Figure 6.11 tests models trained with 1-4 iterations, using 1-10 iterations. We see that models trained with more than one iteration are relatively robust to extra iterations. However, the model trained with just a single iterations gives poor results when many iterations are used for evaluation.

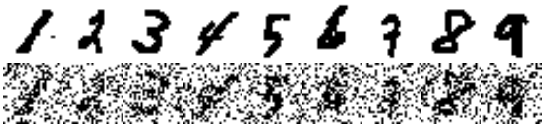
6.5 StreetScenes

The second dataset, known as StreetScenes[24], consists of hand-labeled color outdoor images of streets, reduced to 60x80 resolution. Here, only five labels are used: building, sky, car, road/sidewalk, and tree. There are significant unlabeled regions in the images, which prevents the use of LBP or convex likelihood based learning. (Technically, the conditional pseudolikelihood also cannot be used, but there is an obvious trick of simply using the sum over *observed* variables in Eq. 3.20.)

Because the system only uses linear features, these must be created by hand to

True

50% Noise



Fixed Entropy

A: $w(\mathbf{y}_c) = 1, w(y_i) = .01$

B: $w(\mathbf{y}_c) = 1, w(y_i) = .1$

C: $w(\mathbf{y}_c) = 1, w(y_i) = 1$

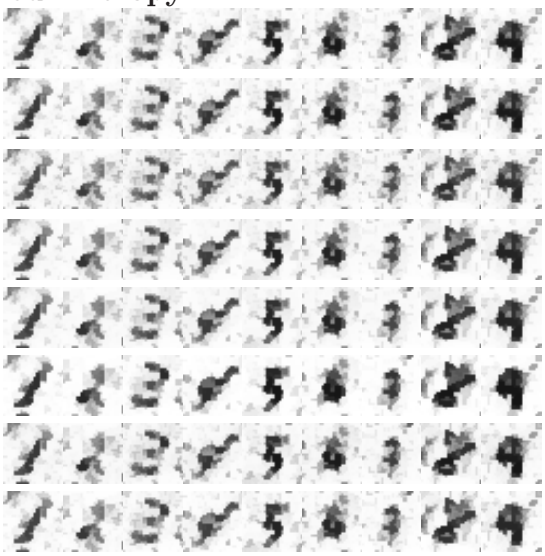
r1: random entropy 1

r2: random entropy 2

r3: random entropy 3

r4: random entropy 4

r5: random entropy 5



Fit Entropy, Initialized to Above Solution

A: $w(\mathbf{y}_c) = 1, w(y_i) = .01$

B: $w(\mathbf{y}_c) = 1, w(y_i) = .1$

C: $w(\mathbf{y}_c) = 1, w(y_i) = 1$

r1: random entropy 1

r2: random entropy 2

r3: random entropy 3

r4: random entropy 4

r5: random entropy 5

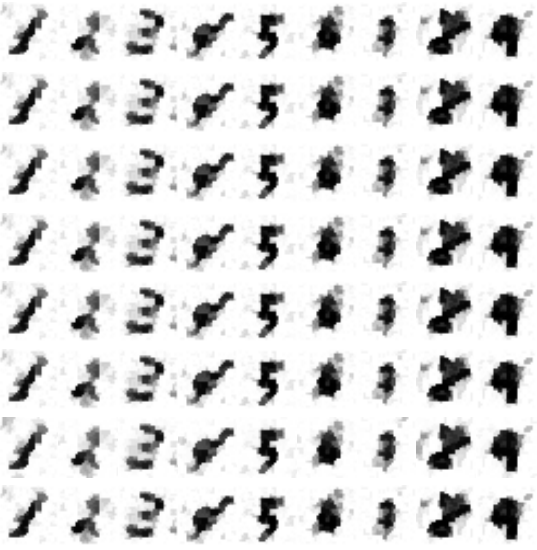


Figure 6.9: Entropy fitting results.

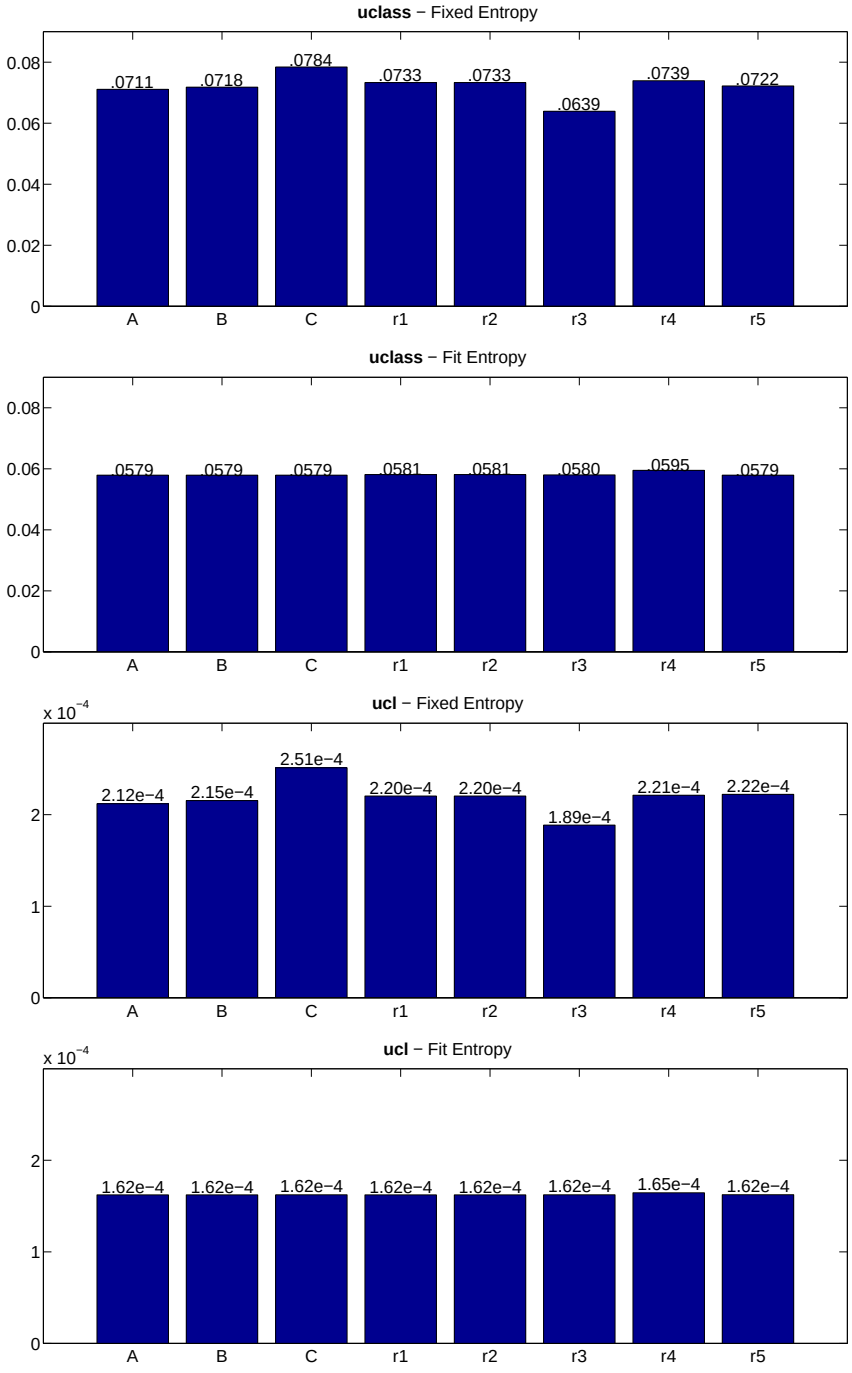


Figure 6.10: Entropy fitting errors. See Figure 6.9 for a key for labels.

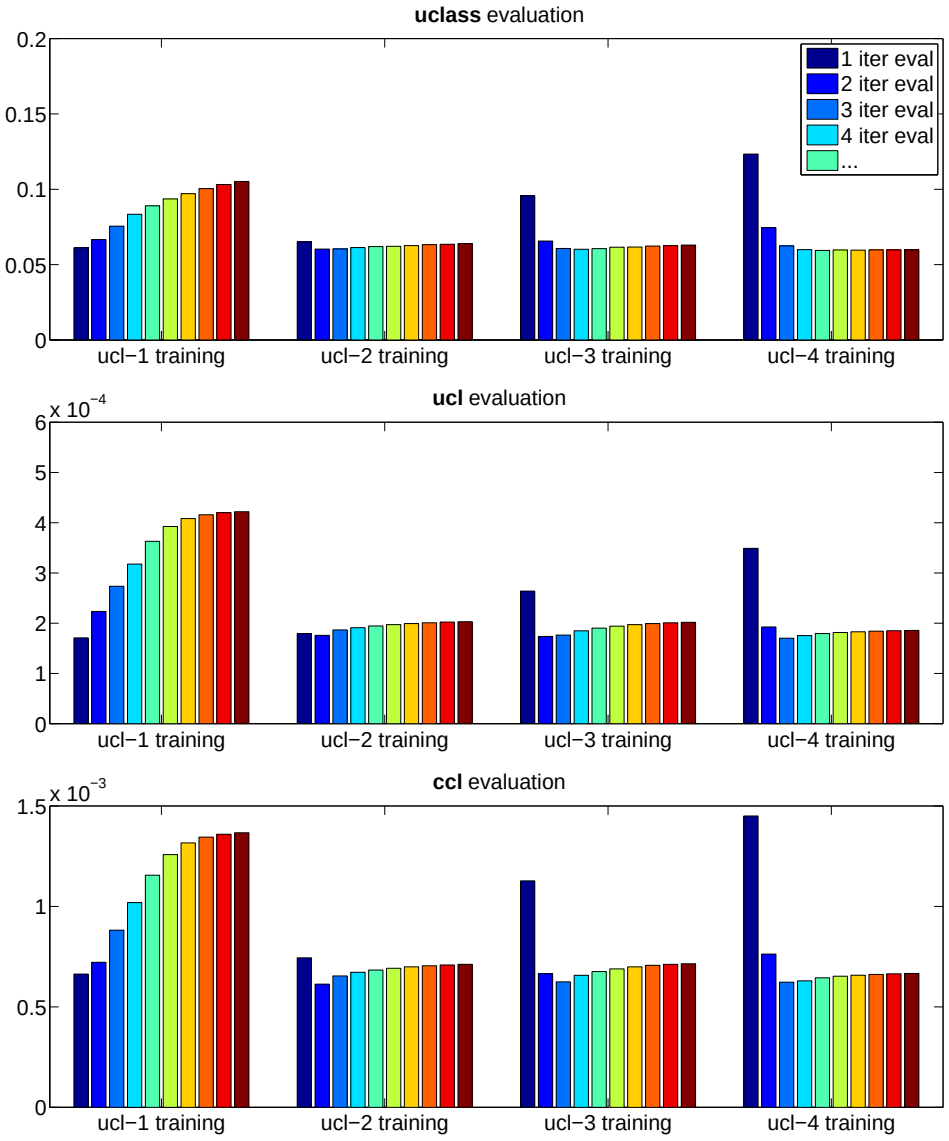


Figure 6.11: Procedurally fit CRFs with varying numbers of iterations.

achieve any reasonable performance. Those used here are summarized in Table 6.2.

- The **raw RGB intensities** are simply the original input image intensities in the three color channels, rescaled to the range $[0, 1]$.

- The **histograms of gradients**[25] at scale 0 are computed by first measuring the gradient on the image. The horizontal derivative dx at scale 1 is approximated by convolving the image with a filter $[-1\,0\,1]$. (At scale k one simply enlarges this filter by substituting a matrix of size $1 + 2k$ for each value.) The vertical derivative dy is approximated with the transpose of this filter. The angle θ at each pixel is given by quantizing $\tan^{-1}(dy/dx)$ to one of 8 values, while the length r is given by $\sqrt{dx^2 + dy^2}$. The feature vector at scale 1 is given by a histogram of the angles θ_i in the surrounding 7x7 patch, with each pixel is weighted by $r_i/\sqrt{\epsilon^2 + \sum_j r_j^2}$, where the sum is over the pixels j in the neighborhood. The constant $\epsilon = .01$ prevents regions with very low gradients from contributing much. At scale k, the same thing is done, where the 7x7 patch is instead taken with a stride of 2^k between the pixels.

- The **cluster indicator functions** were computed by first taking a large sample of randomly selected 5x5 patches, and creating 50 clusters by k-means. The vector at each pixel is simply a vector of zeros with a one for the cluster center closest to the image patch. At half scale, the vector is created by filtering the image, and then taking the 5x5 patch with a stride of 2. At quarter scale, the same thing is done with more filtering, and a stride of 4.

A constant feature was not included, as it would be linearly dependent on the cluster indicator functions (which always sum to one), and thus provide no advantage.

The results here use 100 training and test images at a resolution of 60x80. Because of the computational expense of training this model, only the univariate quadratic loss is used for training. Table 6.3 and Figure 6.12 give the results on test data.

Feature	Number
Raw RGB intensities	3
Histograms of Gradients (Scale 0)	8
Histograms of Gradients (Scale 1)	8
Histograms of Gradients (Scale 2)	8
Cluster indicator functions	50
Cluster indicator functions (half scale)	50
Cluster indicator functions (quarter scale)	50
Total	**177**

Table 6.2: Features used with the StreetScenes dataset.

Method \ Loss	`uclass`	`uquad`
`pseudo`	.847	.0000958
`baseline`	.333	-.0001342
`uquad`	.287	-.0001548
`uquad + ent.fitting`	.239	-.0001823
`uquad-1`	.252	-.0001324
`uquad-2`	.264	-.0001612

Table 6.3: Test errors on the StreetScenes dataset

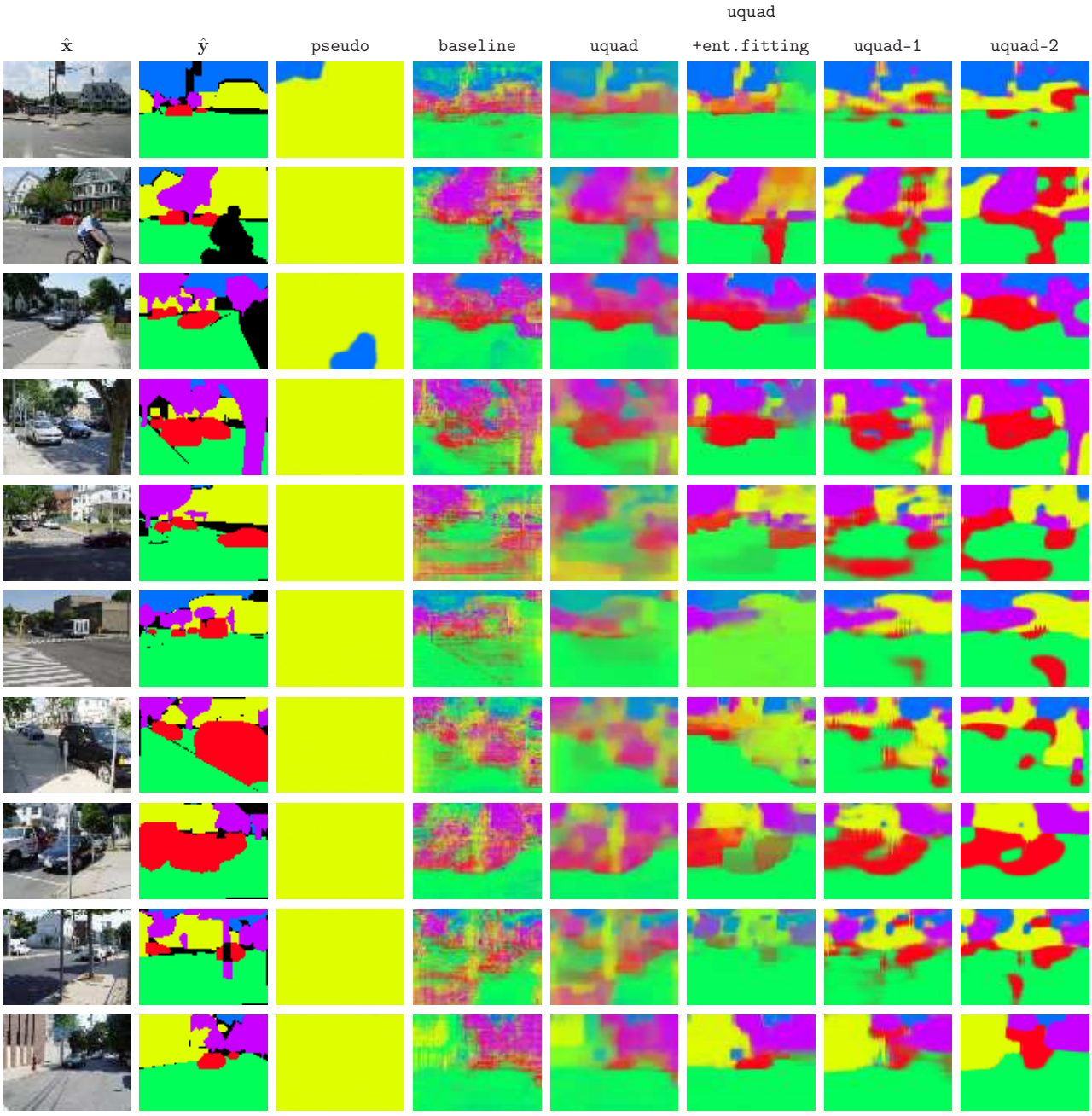


Figure 6.12: StreetScenes results for every tenth image in the test set.

The `baseline` method refers to training a classifier on a totally disconnected graph– i.e. with each pixel predicting its label independently. Surprisingly, pseudolikelihood training performs far worse than the baseline, giving high probability to the label "building" for all pixels. Because of this poor performance, it is preferable to simply initalize parameters by $w = v = 0$. Again, here, procedural fitting is able to outperform implicit fitting.

Chapter 7

Discussion

7.1 Comparison to traditional approaches

In the experiments, both pseudolikelihood and loopy belief propagation based learning performed very poorly on the more difficult problems. (Note, however, that since LBP is based on a non-convex optimization, better performance might be possible though different initialization, update orders, damping heuristics, etc.)

The Convex approximate likelihood performed well, when it could be applied. In general, implicit fitting with a fixed entropy gave somewhat lower errors, while implicit fitting with fit entropies, and procedural fitting did better still. On binary digits with 50% noise, univariate classification rates were as follows.

Method	`uclass` error
`convex`	.072
`ucl` (Implicit Fitting)	.071
`ucl` + entropy fitting (Implicit Fitting)	.059
`ucl-4` (Procedural Fitting, 4 iterations)	.060

There are two other advantages worth mentioning. First, the proposed methods deal easily with hidden variables, simply by taking the sums in the univariate or clique-wise loss functions only over the observed variables. Maximum likelihood learning in graphical models requires techniques such as Expectation-Maximization, with an increased computational cost. So far as the author is aware, use of Expectation-Maximization in the case of approximate inference remains heuristic.

The second advantage is speed. Implicit fitting has no speed advantage over convex likelihood based learning. (Indeed, here the two methods use the same inference algorithm.) However, procedural fitting is far faster. Running a one iteration of LBP on a pairwise connected graph with P pairs of variables, each of which can take L labels, will take $\Theta(P \cdot L^2)$ time (the same time belief propagation takes for exact inference on a singly-connected graph). This is an advantage again the in prediction stage. Standard inference algorithms take varying numbers of iterations, while procedurally fit models take a fixed number by definition.

7.2 Scaling to Huge Problems

Procedural fitting scales linearly in the number of variables, and thus is applicable to very large problems. The only potential problem is the memory requirements– every intermediate message update must be kept in memory in order to backpropagate errors and compute derivatives. This linear scaling may be an issue in situations where many iterations are used. However, there are techniques to reduce the memory requirements of automatic differentiation through storing only a subset of values, and recomputing others as needed [26].

Implicit fitting faces a more serious scaling problem, namely solving increasingly large sparse linear systems. There are two different systems that need to be solved: Step 3(c) of the belief optimization algorithm (Alg. 1), and Step 4 of the algorithm

to calculate $dL/d\theta$ (Alg. 3). However, both of these involve solving systems of the form $(AD^{-1}A^T)^{-1}b$, for a sparse constraint matrix A, and a diagonal matrix D. In all experiments here, these were solved by direct methods, i.e. by factoring the matrix $AD^{-1}A^T$, and then finding the solution by back-substitution. However, the number of nonzeros in the factors scales super-linearly. The obvious solution to scaling these to large problems would be using iterative methods to solve these systems. Some preliminary experimentation was not able to consistently solve these systems in less time than direct methods, unless the system is extremely large (so that neither method is really practical). Most likely, this can be improved through careful preconditioning. An attractive alternative would be the use of algebraic multi-grid solvers.

7.3 Convexity

One major disadvantage of marginal-based loss functions is that they are non-convex, even with favorable assumptions (e.g. exact inference or linear features). The debate about the importance of convexity in loss functions goes back decades, and will not be settled here. When a convex loss function exists, it is often a good idea to initialize the non-convex loss to that solution. This was done in Chapter 3 for exact inference (initializing to the conditional likelihood), and for all loss functions on binary digits in Chapter 6. Note, however, that this is not guaranteed to give the best results. On the StreetScenes dataset, for example, the pseudolikelihood parameters would be a poor initialization.

Note also that methods for training likelihood losses with hidden variables (e.g. Expectation Maximization) use non-convex loss functions, so there is no disadvantage in that case.

7.4 Approximate inference vs. simple models

Rather than assuming that approximate inference is unavoidable, and trying to cope with the approximations, a natural idea is to fit a simpler model. That is, one might deal with tractability at the *modeling* stage, rather than in inference.

Domingos [27] suggested taking the complexity of inference into account while learning. Lowd and Domingos[28] demonstrate an algorithm for learning arithmetic circuits (an alternative representation of a Bayesian network due to Darwiche [29]). The learned compact arithmetic circuits achieve comparable results to Bayesian networks, with huge increases in inference speed.

Other related work includes learning mixtures of tractable distributions, such as fully factorized[30] or tree structured[31].

7.5 Related Work: Energy Based Models

One area of related work is Energy Based Models [32], and related methods, such as Max-Margin Markov networks [33]. These similarly consider graphical models in terms of the energy function that they create. Learning takes place by shaping that objective function such that the minima give good predictions. The major difference is that Energy Based Models focus on MAP, rather than marginal inference. There are two major consequences of this.

The foremost is that MAP and marginal inference represent different priorities (Section 2.2). Disregarding approximations, MAP inference seeks the single joint configuration $\mathbf{y}$ with maximum probability $p(\mathbf{y}|\mathbf{x})$ given the input $\mathbf{x}$. Marginal inference, meanwhile, is concerned with the marginal probabilities $p(y_i|\mathbf{x})$ or $p(\mathbf{y}_c|\mathbf{x})$. Which of these is most appropriate depends on the situation. Roughly speaking, marginal inference is better when we want the most individual variables y_i to be correct, while MAP inference is better when we only care about joint accuracy.

Paradoxically, however, it is common in computer vision to use MAP inference in situations in which one is concerned with univariate accuracy. This tradition goes back at least as far as Geman and Geman's famous 1984 paper [34], in which simulated annealing is used to denoise images, and continues to much present day work using more advanced methods for, e.g., predicting stereo disparity [35].

One can design learning methods to maximize the univariate accuracy of these MAP predictors. However, from a certain theoretical standpoint, this is a strange thing to do. If we have some observation $\mathbf{x}$, and we know the true distribution $p_0(\mathbf{y}|\mathbf{x})$, the optimal inference procedure is Maximum Posterior Marginal Inference– compute the marginals $p_0(y_i|\mathbf{x})$, and then maximize each independently. Thus, if one has plentiful training data, a well-specified model, and exact inference, marginal based learning and inference is guaranteed to give optimal predictions, while an EBM based on the same graphical model is not.

Yet from more practical standpoint, training an EBM for univariate accuracy is perfectly reasonable, and could outperform a marginalization approach in practice. The reason is that the graphical model may not be well-specified, and so the above guarantees do not apply. Absent assumptions about the true distribution, MAP and marginal inference are just different classes of functions mapping from input to predictions, neither of which is superior in general. Indeed, the probabilistic interpretation of these models is sometimes explicitly downplayed [36, section 3.2], or not even mentioned[37].

The second difference is the possibility of accounting for approximate inference in learning. It is relatively easy to account for approximations in marginal inference. One replaces the true entropy and marginal polytope with approximations which then (if convex) can be optimized exactly. This chapter has focused on how to "fit around the approximations" during learning. For MAP inference, however, it is not clear how to analytically account for approximations. In certain restricted graphs

(e.g. low treewidth graphs, or binary associative networks [38][36, section 8]), exact inference is possible. In general graphs, these methods replace exact inference with approximations, and so success is not experimental. In practice, this often works well. (Further, Taskar et al. [36, section 7.2]) conjecture that replacing exact inference with approximate does indeed result in approximation-aware learning, to some degree.)

7.6 Probabilistic Modeling and Model Error

This thesis has argued for fitting graphical models for marginal accuracy and then using marginal inference for predictions. It should be noted that this has two supporting arguments, with somewhat contradictory premises.

- **Optimality of marginal inference.** The first part of this argument is that if one is interested in maximum univariate error, the optimal inference procedure is to compute the marginals $p(y_i|\mathbf{x})$, and then find the label y_i maximizing each independently.

- **Robustness to model error.** The second part of this argument is that univariate or clique-wise loss functions are preferable to likelihood based loss functions because of model errors.

These two arguments are not simultaneously rigorous. In order to know that marginal inference is optimal, we must assume that the model is well-specified. However, marginal-based loss functions are preferable precisely when the model is *misspecified*.

Thus, this argument is heuristic in assuming that marginal inference continues to be optimal in the case of model misspecification. This seems reasonable in the case of only small model error. If the model is very wrong, however, this need not be true.

There is evidence that the models used in practice are often misspecified. For example, the pseudolikelihood frequently gives very poor results, even when data is

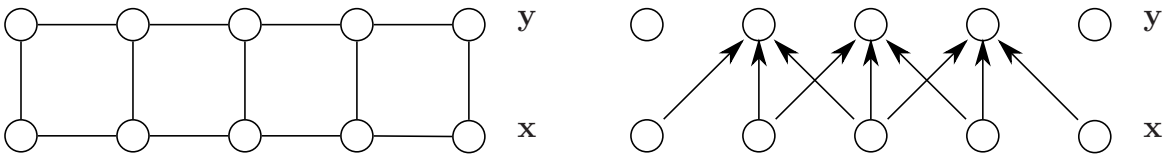


Figure 7.1: A CRF vs. direct prediction of marginals.

plentiful. This is a sign that the model is not well-specified. If if one is not serious about making conditional independence assumptions, it might be better to disregard the probabilistic interpretation of graphical models.

7.7 Why fit a joint distribution just to marginalize?

The goal here is the prediction of accurate marginals. Why, then, do we fit joint distributions? We have to work very hard to fit $p(\mathbf{y}|\mathbf{x})$, just to marginalize down to $p(y_i|\mathbf{x})$. (Or, rather, *approximately* marginalize.) Why not "eliminate the middleman", and simply fit $p(y_i|\mathbf{x})$ directly?

Let us be more precise. Consider the CRF on the left of Fig. 7.1. Rather than fitting a joint distribution $p(\mathbf{y}|\mathbf{x})$ and then marginalizing, one might identify a set of "parents" $\mathbf{x}_{\pi(i)}$ for each variable y_i, and directly fit $p(y_i|\mathbf{x}_{\pi(i)})$ using any standard supervised learning method (e.g. logistic regression, k-nearest neighbors, neural networks, etc.). How will these predicted marginals compare to those obtained from a graphical model approach?

In many cases, this will work very well. Given how comparatively easy this approach is, the practitioner should probably try this before resorting to a graphical model. In other cases, however, this will not succeed. the trouble lies in the size of the set $\pi(i)$. If there exists a small set such that y_i is conditionally independent of $x_j, x \notin \pi(i)$ given $\mathbf{x}_{\pi(i)}$, then the approach will work well. Often, however, this set might need to be large, and the mapping $p(y_i|\mathbf{x}_{\pi(i)})$ very complex, when a very simple CRF would work well. Notice that in the CRF in Fig. 7.1, y_i will usually be

dependent on the entire vector $\mathbf{x}$.

Bibliography

[1] Dan Roth. On the hardness of approximate reasoning. *Artificial Intelligence*, 82(1-2):273 – 302, 1996.

[2] Venkat Chandrasekaran, Nathan Srebro, and Prahladh Harsha. Complexity of inference in graphical models. In *Proceedings of the 24th Annual Conference on Uncertainty in Artificial Intelligence (UAI-08)*, null, 2008. null.

[3] Julian Besag. Spatial interaction and the statistical analysis of lattice systems. *Journal of the Royal Statistical Society. Series B (Methodological)*, 36(2):192–236, 1974.

[4] J L Marroquin. *Probabilistic solution of inverse problems*. PhD thesis, Massachusetts Institute of Technology, 1985.

[5] David J. C. MacKay. *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press, 2003. Available from `http://www.inference.phy.cam.ac.uk/mackay/itila/`.

[6] Jonathan S. Yedidia, William T. Freeman, and Yair Weiss. Constructing free energy approximations and generalized belief propagation algorithms. *IEEE Transactions on Information Theory*, 51:2282–2312, 2004.

[7] M. J. Wainwright and M. I. Jordan. Graphical models, exponential families, and variational inference. *Foundations and Trends in Machine Learning*, 1:1–305, 2008.

[8] Vladimir N. Vapnik. *The Nature of Statistical Learning Theory*. Springer, November 1999.

[9] T. Minka. Divergence measures and message passing (MSR-TR-2005-173). Technical report, Microsoft Research, 2005.

[10] Julian Besag. Statistical analysis of non-lattice data. *The Statistician*, 24(3):179–195, 1975.

[11] Julian Besag. Efficiency of pseudolikelihood estimation for simple gaussian fields. *Biometrika*, 64(3):616–618, 1977.

[12] Percy Liang and Michael Jordan. An asymptotic analysis of generative, discriminative, and pseudolikelihood estimators. In *ICML*, 2008.

[13] J. Rissanen. Lectures on statistical modeling theory. `http://www.mdl-research.org/pub/lectures.pdf`, 2008.

[14] L Breiman. Statistical modeling: The two cultures. *Statistical Science*, 16(3):199–215, 2001.

[15] Sham Kakade, Yee Whye Teh, and Sam T. Roweis. An alternate objective function for markovian fields. In *ICML*, pages 275–282, 2002.

[16] J Domke. Learning convex inference of marginals. In *Uncertainty in Artificial Intelligence*, 2008.

[17] Samuel S. Gross, Olga Russakovsky, Chuong B. Do, and Serafim Batzoglou. Training conditional random fields for maximum labelwise accuracy. In

B. Schölkopf, J. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems 19*, pages 529–536. MIT Press, Cambridge, MA, 2007.

[18] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, March 2004.

[19] T Heskes. Convexity arguments for efficient minimization of the bethe and kikuchi free energies. *Journal of Artificial Intelligence Research*, 26:153–190, 2006.

[20] V Ganapathi, D Vickrey, J Duchi, and D Koller. Constrained approximate maximum entropy learning of markov random fields. In *Uncertainty in Artificial Intelligence*, 2008.

[21] D. Gay. Semiautomatic differentiation for efficient gradient computations. Technical report, Sandia National Laboratories, 2004.

[22] M. J. Wainwright, T. Jaakkola, and A. S. Willsky. Tree-reweighted belief propagation and approximate ml estimation by pseudo-moment matching. In *Workshop on Artificial Intelligence and Statistics*, 2003.

[23] Thomas Minka. Expectation propagation for approximate bayesian inference, 2001.

[24] Stanley Michael Bileschi. *StreetScenes: Towards Scene Understanding in Still Images*. PhD thesis, Massachusetts Institute of Technology, 2006.

[25] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In Cordelia Schmid, Stefano Soatto, and Carlo Tomasi, editors, *International Conference on Computer Vision & Pattern Recognition*, volume 2, pages 886–893, INRIA Rhône-Alpes, ZIRST-655, av. de l'Europe, Montbonnot-38334, June 2005.

[26] Andreas Griewank. Achieving logarithmic growth of temporal and spatial complexity in reverse automatic differentiation. *Optimization Methods and Software*, 1:35–54, 1992.

[27] Pedro Domingos. Structured machine learning: Ten problems for the next ten years. *Machine Learning*, 73:3–23, 2008.

[28] Daniel Lowd and Pedro Domingos. Learning arithmetic circuits. In *Uncertainty in Artificial Intelligence*, 2008.

[29] A Darwiche. A differential approach to inference in bayesian networks. *Journal of the ACM*, 50:280–305, 2003.

[30] Petri Kontkanen, Petri Myllymäki, and Henry Tirri. Constructing bayesian finite mixture models by the em algorithm. Technical report, University of Helsinki, Department of Computer Science, 1996.

[31] M Meila and M Jordan. Learning with mixtures of trees. *Journal of Machine Learning Research*, 1:1–48, 2000.

[32] Yann LeCun, Sumit Chopra, Raia Hadsell, Ranzato Marc'Aurelio, and Fu-Jie Huang. A tutorial on energy-based learning. In G. Bakir, T. Hofman, B. Schölkopf, A. Smola, and B. Taskar, editors, *Predicting Structured Data*. MIT Press, 2006.

[33] B. Taskar, C. Guestrin, and D. Koller. Max–margin Markov networks. In *Advances in Neural Information Processing Systems*, 2004.

[34] Stuart Geman and Donald Geman. Stochastic relaxation, gibbs distributions and the bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6(6):721–741, November 1984.

[35] Richard Szeliski, Ramin Zabih, Daniel Scharstein, Olga Veskler, Vladimir Kolmogorov, Aseem Agarwala, Marshall Tappen, and Carsten Rother. A comparative study of energy minimization methods for markov random fields with smoothness-based priors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(6), 2008.

[36] B. Taskar, C. Guestrin, V. Chatalbashev, and D. Koller. Max-margin markov networks. *Journal of Machine Learning Research*, To appear.

[37] Yuri Boykov, Olga Veksler, and Ramin Zabih. Fast approximate energy minimization via graph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23:2001, 1999.

[38] Vladimir Kolmogorov and Ramin Zabih. What energy functions can be minimized via graph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26:65–81, 2004.