1 Training Structured Predictors Through Iterated Logistic Regression

Justin Domke justin.domke@nicta.com.au
NICTA & The Australian National University
Canberra, Australia

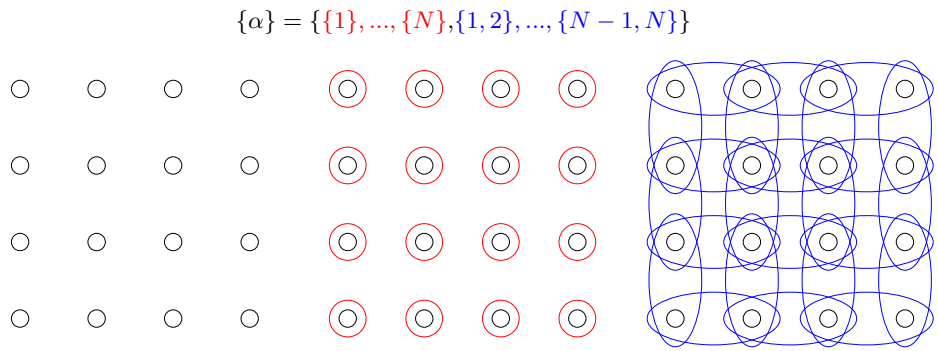
In a setting where approximate inference is necessary, structured learning can be formulated as a joint optimization of inference "messages" and local potentials. This chapter observes that, for fixed messages, the optimization problem with respect to potentials takes the form of a logistic regression problem, biased by the current set of messages. This observation leads to an algorithm that alternates between message-passing inference updates, and learning updates. It is possible to employ any set of potential functions where an algorithm exists to optimize a logistic loss, including linear functions, boosted decision trees, and multi-layer perceptrons.

1.1 Introduction

This chapter is concerned with the discrete structured prediction problem, in which some input vector x is used to predict an output vector y by maximizing an "energy" function F to find

$$y^* = \operatorname*{argmax}_{y \in \mathcal{Y}} F(x, y). \tag{1.1}$$

Here, $x \in \mathcal{X}$ is the input space, typically a set of real vectors. The output space is a set of N dimensional vectors $y \in \mathcal{Y} = \mathcal{Y}_1 \times \mathcal{Y}_2 \times ... \times \mathcal{Y}_N$, where $\mathcal{Y}_i$ is a discrete set of the values y_i can obtain.



(a) Original variables (b) Univariate regions (c) Pair regions

Figure 1.1: In imaging problems, it is common to use a "grid" structure, where there is one region $\alpha = \{i\}$ corresponding to each pixel i, and one region $\alpha = \{i, j\}$ corresponding to each neighboring pair in the 4-connected grid.

Here, we assume that the function F decomposes as

$$F(x, y) = \sum_{\alpha} f_\alpha(x, y_\alpha). \tag{1.2}$$

This sum ranges over a set of *regions* α (Koller and Friedman, 2009, Sec 11.3.7.3), each of which is a subset of $\{1, 2, ..., N\}$. Each region is selected to capture a set of interdependent output variables, see Figure 1.1. The function f_α encourages the variables y_α to take on a value such that $f_\alpha(x, y_\alpha)$ is high.

Given some annotated dataset $(x^1, y^1), ..., (x^M, y^M)$, the structured learning problem is to pick the form of the function F. Intuitively speaking, one would like to select F such that

$$y^k \approx \underset{y \in \mathcal{Y}}{\operatorname{argmax}}\ F(x^k, y) \qquad \forall k \in \{1, ..., M\}. \tag{1.3}$$

However, there are two complicating factors. First, it will typically be impossible to adjust F such that that Eq. 1.3 is exactly equal for all k. (More precisely, using a class of functions with sufficient power to accomplish this would be undesirable due to overfitting.) Thus, a loss function must be specified to trade-off between different types of errors on different examples. Secondly, when the function F changes, the maximizing argument of Eq. 1.3 changes discontinuously. Thus, the loss must be stated *implicitly* in terms of the result of a maximization, which presents computational difficulties in selecting the best function.

1.2 Linear vs. Nonlinear Learning

Most commonly, $F(x, y) = \sum_\alpha f_\alpha(x, y_\alpha)$ can be written in the linear form

$$F(x, y) = \sum_\alpha w^T \Phi_\alpha(x, y_\alpha).$$

The function Φ_α produces a set of features that reveal aspects of the interdependency of the variables y_α, possibly also taking into account the input x. The learning problem is then to select the vector of weights w so as to make the mapping in Eq. 1.1 accurate.

This chapter considers the more general case where each factor f_α is only assumed to be a member of some set of (possibly nonlinear) functions $\mathcal{F}_\alpha$. The learning problem is thus instead to select $f_\alpha \in \mathcal{F}_\alpha$ for all α.

A motivation for using more general function classes is the common existing practice to fit a nonlinear function class to predict each variable independently. These univariate potentials are then fixed, while linear weights are adjusted for interaction potentials (Section 1.9). There are two weaknesses to this approach. Firstly, only linear weights are learned for interaction potentials, rather than more powerful functions. Secondly, the nonlinear functions learned for the univariate potentials are suboptimal for joint prediction.

1.3 Overview

The algorithm presented in this chapter can be seen as building upon two streams of research for fitting structured predictors in the setting where exact inference is intractable:

- Piecewise learning (Sutton and Mccallum, 2009) trains a structured predictor by splitting the model into a set of "pieces", which could be individual factors, or other structures where exact inference can be performed. These pieces are then trained independently of the rest of the graph. This can be justified as a bound on the true likelihood. This has advantages of computational convenience, and is fairly amenable to using nonlinear potential functions when learning, but does not always lead to good performance for joint prediction, since the bound on the likelihood can be quite loose.
- Algorithms based on formulating a joint learning and inference objective (Hazan and Urtasun, 2012; Meshi et al., 2010) deal with approximate inference in a principled way by alternating between message-passing updates, and gradient updates of parameters. However, these algorithms deal only with linear potential functions.

The algorithm presented in this chapter is similar to both of the above. As pictured in Fig. 1.3, it begins by training all factors separately via logistic regression. This is similar to piecewise learning, and it is possible to use nonlinear factors. However, after this first step, message-passing inference proceeds, which creates a new set of logistic regression problems reflecting the biases from other factors. Iterating this process leads to an optimum of a learning objective reflecting both messages and potential functions.

Proofs of all the theorems stated in this chapter are given in the appendix.

1.4 Loss Functions

Structured learning usually follows the standard framework of empirical risk minimization, wherein given a dataset $(x^1, y^1), ..., (x^N, y^N)$, the goal of learning is to select F to minimize the empirical risk

$$R(F) = \sum_k l(x^k, y^k; F), \tag{1.4}$$

for some loss function l. In early work on structured learning (Taskar et al., 2003), the loss takes the form

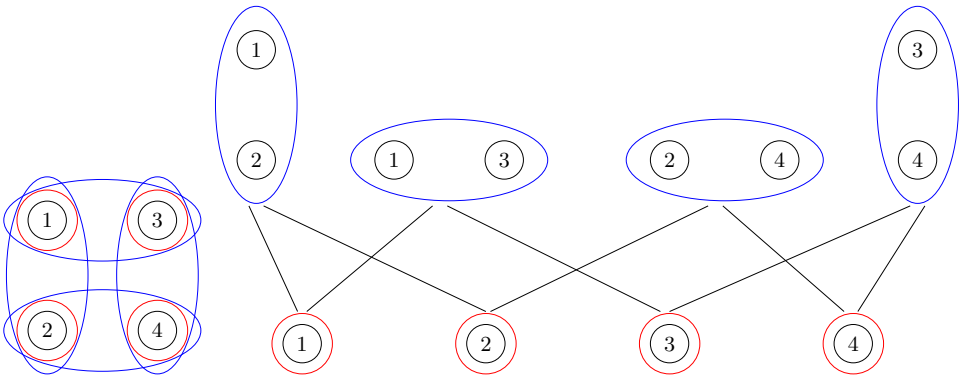
$$l_0(x^k, y^k; F) = -F(x^k, y^k) + \max_{y \in \mathcal{Y}} F(x^k, y) + \Delta(y^k, y), \tag{1.5}$$

where $\Delta(y^k, y)$ is a discrepancy measures of how "different" y^k is from y. If $\Delta = 0$, notice that l_0 is a perceptron-type loss, which measures the energy of the top-scoring value $\max_{y \in \mathcal{Y}} F(x^k, y)$ minus the energy of the true value $F(x^k, y^k)$. The loss will be zero if y^k scores as well as any other value. When Δ is nonzero, it is necessary for the score of y^k to be at least $\Delta(y^k, y)$ better than y in order for the loss to be zero. Essentially, if some configuration y is extremely "bad", then y^k must score significantly higher than y in order to incur zero loss.

A common discrepancy function is the indicator $\Delta(y^k, y) = I[y^k \neq y]$, which is one if $y^k \neq y$ and zero if $y^k = y$. In this case, it is easy to show that the loss becomes the multiclass hinge loss (Crammer and Singer, 2002)

$$l_0(x^k, y^k; F) \quad = \quad \max\left(0, 1 + \max_{y \neq y^k \in \mathcal{Y}} F(x^k, y) - F(x^k, y^k)\right). \tag{1.6}$$

Another common discrepancy (which will be used in the experiments later in this chapter) is the Hamming distance $\Delta(y^k, y) = \sum_i I[y_i^k \neq y_i]$, which measures the number of components in which y^k and y differ. The rest of this chapter will assume that the discrepancy decomposes over the set of regions, i.e. it can be written as $\Delta(y^k, y) = \sum_\alpha \Delta_\alpha(y_\alpha^k, y_\alpha)$.



(a) Original Graph (b) Parent-Child graph

Figure 1.2: An example graph with four nodes and eight regions, namely $\{\alpha\} = \{\{1\},\{2\},\{3\},\{4\},\{1,2\},\{1,3\},\{2,4\},\{3,4\}\}$. The parent child graph represents each region, with links between "parent" regions that contain all the nodes in "child" regions.

The maximization in Eq. 1.5 ranges over all discrete labelings, a set that is in general exponentially large. Thus, this loss is practical only when there exists a special structure that allows one to quickly find the maximum. For example, if the graph obeys a tree structure, then dynamic-programming algorithms can compute the maximum. If using a linear energy, learning can then proceed either through subgradient descent (Ratliff et al., 2007) or constraint generation methods (Tsochantaridis et al., 2005; Taskar et al., 2003).

To overcome these issues, a common approach is to use a relaxed version of the loss, where rather than optimizing over all discrete labelings, one optimizes over all sets of marginals. The relaxed loss is defined as

$$l_1(x^k, y^k; F) = -F(x^k, y^k) + \max_{\mu \in \mathcal{M}} F(x^k, \mu) + \Delta(y^k, \mu). \tag{1.7}$$

Here $\mu = \{\mu_\alpha(y_\alpha)\}$ is a set of marginals, assigning some probability to each possible configuration of each region. As a slight abuse, the notation of F and Δ is overloaded to allow arguments of marginals[1].

If the set $\mathcal{M}$ is defined appropriately, l_1 is equivalent to l_0. Specifically, suppose that each marginal μ_α is restricted to put all probability on one

1. Specifically, F is defined as $F(x^k, \mu) = \sum_\alpha \sum_{y_\alpha} f_\alpha(x^k, y_\alpha)\mu_\alpha(y_\alpha)$, while Δ is defined as $\Delta(y^k, \mu) = \sum_\alpha \sum_{y_\alpha} \Delta_\alpha(y^k_\alpha, y_\alpha)\mu_\alpha(y_\alpha)$.

configuration, and that these assignments are consistent between regions that consider the same set of variables. Let $\mu_{\alpha\beta}(y_\beta) = \sum_{y_{\alpha\setminus\beta}} \mu_\alpha(y_\alpha)$ denote μ_α marginalized out over a subregion β. If $\mathcal{M}$ takes the form

$$\begin{aligned}\mathcal{M} = \{\mu | \mu_\alpha(y_\alpha) \in \{0,1\} \quad & \forall \alpha, y_\alpha, \\ \sum_{y_\alpha} \mu_\alpha(y_\alpha) = 1 \quad & \forall \alpha, \\ \mu_{\alpha\beta}(y_\beta) = \mu_\beta(y_\beta) \quad & \forall \alpha \supset \beta, y_\beta\},\end{aligned} \tag{1.8}$$

and it is not hard to show that l_0 and l_1 are equivalent. (See Chapter **??** for a detailed discussion of the marginal polytope and its relaxations.) Here, a "parent-child" representation of the graph is used, where each region α interacts with those regions that are subsets or supersets (Figure 1.2). However, with this constraint set, l_1 is no easier to evaluate than l_0, as it takes the form of a difficult integer linear programming problem. However, as is commonly done to approximate integer programs (Vazirani, 2001), this can be approximated by replacing $\mathcal{M}$ with a set that makes the maximization in Eq. 1.7 easier. Specifically, it is common (Finley and Joachims, 2008) to use a linear programming relaxation like

$$\begin{aligned}\mathcal{M} = \{\mu | \mu_\alpha(y_\alpha) \in [0,1] \quad & \forall \alpha, y_\alpha, \\ \sum_{y_\alpha} \mu_\alpha(y_\alpha) = 1 \quad & \forall \alpha, \\ \mu_{\alpha\beta}(y_\beta) = \mu_\beta(y_\beta) \quad & \forall \alpha \supset \beta, y_\beta\},\end{aligned} \tag{1.9}$$

which leads to a loss that can be evaluated through the solution of a linear programming problem. Moreover, since $\mathcal{M}$ has been replaced with a larger set, it is easy to see that $l_1 \geq l_0$, meaning good performance on this surrogate loss will guarantee good performance in the original one.

It is convenient to introduce the notation of

$$\theta_F^k(y_\alpha) = f_\alpha(x^k, y_\alpha) + \Delta_\alpha(y_\alpha^k, y_\alpha), \tag{1.10}$$

in which case l_1 can be written in the equivalent form

$$l_1(x^k, y^k; F) = -F(x^k, y^k) + \max_{\mu \in \mathcal{M}} \theta_F^k \cdot \mu, \tag{1.11}$$

where we define the inner product between θ and μ as $\theta \cdot \mu = \sum_\alpha \sum_{y_\alpha} \theta(y_\alpha)\mu_\alpha(y_\alpha)$.

Now, while it is tractable to compute l_1, it is not smooth as a function of F, which rules out the use of certain optimization methods for learning. A solution to this is to add "entropy" smoothing. That is, to replace the loss

with

$$l(x^k, y^k; F) = -F(x^k, y^k) + \max_{\mu \in \mathcal{M}} \left(\theta_F^k \cdot \mu + \epsilon \sum_\alpha H(\mu_\alpha) \right), \tag{1.12}$$

where $H(\mu_\alpha) = -\sum_{y_\alpha} \mu_\alpha(y_\alpha) \log \mu_\alpha(y_\alpha)$. Hazan and Urtasun (2012) consider a more general class of approximate entropies (where different factors α can have varying weights ϵ_α), and note that depending on the entropy approximation and the divergence Δ, l_2 can encompass both surrogate likelihood (Wainwright, 2006) and structured prediction types of objectives.

Meshi et al. (2012) consider approximating the inference problem of $\max_{\mu \in \mathcal{M}} \theta \cdot \mu$ with the smoothed problem $\max_{\mu \in \mathcal{M}} \theta \cdot \mu + \epsilon \sum_\alpha H(\mu_\alpha)$. They show that message-passing algorithms for performing this maximization can have guaranteed convergence rates and that difference of the two objectives is linear in ϵ. A similar result can be shown that bounds the difference of the two losses, as stated in the following theorem.

Theorem 1.1. *l and l_1 are bounded by (where $|y_\alpha| = \prod_{i \in \alpha} |\mathcal{Y}_i|$ is the number of configurations of y_α)*

$$l_1(x, y, F) \leq l(x, y, F) \leq l_1(x, y, F) + \epsilon H_{\max}, \quad H_{\max} = \sum_\alpha \log |y_\alpha|.$$

It is sometimes convenient to write l as

$$\begin{aligned} l(x^k, y^k; F) &= -F(x^k, y^k) + A(\theta_F^k), \\ A(\theta) &= \max_{\mu \in \mathcal{M}} \theta \cdot \mu + \sum_\alpha \epsilon H(\mu_\alpha). \end{aligned}$$

Intuitively speaking, here $F(x^k, y^k)$ measures the score of the correct output y^k. Meanwhile, $A(\theta_F^k)$ measures the score of the "worst" configuration, taking into account the discrepancy Δ and approximated using entropy smoothing and the relaxation of $\mathcal{M}$ from Eq. 1.9. Thus, one would like $F(x^k, y^k)$ to be large, and $A(\theta_F^k)$ to be small, which is exactly what l measures.

1.5 Message Passing Inference

Evaluating the loss defined in the previous section on a particular datum requires performing an optimization to evaluate $A(\theta_F^k)$ for that datum. As will be discussed in the following section, it is convenient to represent A in

a dual form as a minimization.

Theorem 1.2. *$A(\theta)$ can be represented in the dual form $A(\theta) = \min_\lambda A(\lambda, \theta)$, where*

$$A(\lambda,\theta) = \max_{\mu\in\mathcal{N}} \theta\cdot\mu + \epsilon\sum_\alpha H(\mu_\alpha) + \sum_\alpha \sum_{\beta\subset\alpha} \sum_{x_\beta} \lambda_\alpha(x_\beta)\left(\mu_{\alpha\beta}(y_\beta) - \mu_\beta(y_\beta)\right), \tag{1.13}$$

and $\mathcal{N} = \{\mu | \sum_{y_\alpha} \mu_\alpha(y_\alpha) = 1\, \forall\alpha, \mu_\alpha(y_\alpha) \geq 0\, \forall\alpha, y_\alpha\}$ is the set of locally normalized pseudomarginals. Moreover, for a fixed λ, the maximizing μ is given by

$$\mu_\alpha(y_\alpha) = \frac{1}{Z_\alpha}\exp\left(\frac{1}{\epsilon}\left(\theta(y_\alpha) + \sum_{\beta\subset\alpha}\lambda_\alpha(y_\beta) - \sum_{\gamma\supset\alpha}\lambda_\gamma(y_\alpha)\right)\right), \tag{1.14}$$

where Z_α is a normalizing constant to ensure that $\sum_{y_\alpha}\mu_\alpha(y_\alpha) = 1$. Moreover, the actual value of $A(\lambda,\theta)$ is

$$A(\lambda,\theta) = \sum_\alpha \epsilon\log\sum_{y_\alpha}\exp\left(\frac{1}{\epsilon}\left(\theta(y_\alpha) + \sum_{\beta\subset\alpha}\lambda_\alpha(y_\beta) - \sum_{\gamma\supset\alpha}\lambda_\gamma(y_\alpha)\right)\right). \tag{1.15}$$

The problem remains of how to actually minimize $A(\lambda,\theta)$ with respect to λ. This is a smooth unconstrained optimization, which could in principle be performed by a variety of generic optimization methods, for example gradient descent. However, there is now long precedent for optimizing objectives like this through "message-passing" algorithms that more closely mirror the structure of the graph (Wainwright and Jordan, 2008).

Consider doing coordinate descent. Danskin's theorem states that taking the derivative of $A(\lambda,\theta)$ with respect to an element $\lambda_\alpha(y_\beta)$ will recover the constraint that this multiplier enforces, namely that

$$\frac{dA(\lambda,\theta)}{d\lambda_\alpha(y_\beta)} = \mu_{\alpha\beta}(y_\beta) - \mu_\beta(y_\beta),$$

where μ is as defined in Eq. 1.14.

Thus, if one can find a set of multipliers such that all marginalization constraints are satisfied, the gradient of $A(\lambda,\theta)$ with respect to λ is zero, meaning the global optimum would have been found. In practice, such a solution cannot usually be found in closed-form, and one resorts to iterative

methods. Suppose, however, that one could adjust the values of a subset of multipliers $\lambda_\alpha(x_\beta)$ to enforce the corresponding set of constraints, while leaving other multipliers fixed. This would mean $A(\lambda, \theta)$ is optimal in the adjusted multipliers, for the fixed values of the non-adjusted multipliers. Thus, an iterative process that repeatedly adjusts blocks of multipliers like this constitutes a block coordinate descent scheme.

Different sizes of "blocks" are possible, ranging from a single set of multipliers from a region α to a subregion β, to subtrees of the original graph (Sontag and Jaakkola, 2009). Here, we consider an intermediate strategy, where, given some region ν, all multipliers $\lambda_\alpha(y_\nu)$, with $\alpha \supset \nu$ are adjusted simultaneously. This is essentially what is done in the parent-child algorithm (Heskes, 2006) and the "star" update of Meshi et al. (2012), albeit with slightly less general conditions on the graph structure than used here.

Theorem 1.3. *Suppose that, for all $\eta \supset \nu$ simultaneously, we set $\lambda'_\eta(y_\nu) = \lambda_\eta(y_\nu) + \delta_\eta(y_\nu)$, where*

$$\delta_\eta(y_\nu) = \frac{\epsilon}{1+N_\nu}\left(\log \mu_\nu(y_\nu) + \sum_{\eta' \supset \nu} \log \mu_{\eta'\nu}(y_\nu)\right) - \epsilon \log \mu_{\eta\nu}(y_\nu). \quad (1.16)$$

and $N_\nu = |\{\eta | \eta \supset \nu\}|$. Then, if μ' denotes the marginals after update, the marginalization conditions $\mu'_{\eta\nu}(y_\nu) = \mu'_\nu(y_\nu)$ will hold. Moreover, each will be proportional to the geometric mean of all marginals considered, i.e.

$$\mu'_{\eta\nu}(y_\nu) = \mu'_\nu(y_\nu) \propto \left(\mu_\nu(y_\nu) \prod_{\eta' \supset \nu} \mu_{\eta'\nu}(y_\nu)\right)^{1/(1+N_\nu)}, \quad (1.17)$$

1.6 Joint Learning and Inference

Explicitly writing the problem of minimizing the empirical risk (Eq. 1.4) with respect to F using the final loss (Eq. 1.12) gives the optimization of

$$\begin{aligned}\min_F R(F) &= \min_F \sum_k \left[-F(x^k, y^k) + A(\theta^k_F)\right] \\ &= \min_F \sum_k \left[-F(x^k, y^k) + \max_{\mu \in \mathcal{M}} \left(\theta^k_F \cdot \mu + \epsilon \sum_\alpha H(\mu_\alpha)\right)\right],\end{aligned}$$

which takes the form of a saddle-point problem, since one is minimizing with

respect to F, but maximizing with respect to all the inference variables. One could conceivably solve this through an algorithm for direct saddle-point optimization (Nedi and Ozdaglar, 2009), but this is less convenient than a joint minimiztion.

However, as shown in the previous section, $A(\theta)$ has a dual representation as $A(\theta) = \min_\lambda A(\lambda, \theta)$. Thus, one can instead write the problem of minimizing the empirical risk as

$$\min_F R(F) = \min_F \min_{\{\lambda^k\}} \sum_k \left[-F(x^k, y^k) + A\left(\lambda^k, \theta_F^k\right) \right], \tag{1.18}$$

where λ^k is the vector of messages corresponding to datum k. Meshi et al. (2010) pursue an optimization like Eq. 1.18, though without entropy smoothing. They learn linear weights through iteratively updating λ^k for a single datum k, and then taking a stochastic gradient step with respect to weights. Similarly, but including entropy smoothing, Hazan and Urtasun (2012) learn linear weights, alternating between message-passing updates to $\{\lambda^k\}$, and gradient updates to weights.

This chapter builds on this previous work in two ways. As before, optimization alternates between updating the energy F and the messages λ. However, here it is observed that, given a fixed set of messages, the problem of optimizing F with respect to an individual factor f_α is equivalent to solving a logistic regression problem with "bias" terms determined by the current messages added to each datum. There are two possible reasons such an observation might be useful. First, one can optimize the empirical risk "all the way" for fixed messages, rather than taking a single gradient step. This allows one to use a range of standard optimization methods, possibly speeding up convergence. Second, it is possible to use nonlinear functions f_α, provided only that some algorithm exists to minimize a logistic loss with respect to that function class. Thus, one can easily use, e.g., decision trees in structured prediction, with no need to develop new specialized learning algorithms.

1.7 Logistic Regression

In traditional linear logistic regression, one is given a dataset $(x^1, y^1), ..., (x^N, y^N)$, where $x^k \in \mathcal{R}^N$, and $y^k \in \{1, 2, ..., L\}$. Then, the logistic regression optimization is to find

$$\max_W \sum_k \left[(Wx^k)_{y^k} - \log \sum_y \exp(Wx^k)_y \right].$$

Here, $(Wx)_y$ denotes the y-th component of the vector of "margins" Wx. One interpretation of this optimization is fitting a conditional likelihood with a probabilistic model of the form $p(y|x;W) \propto \exp(Wx)_y$ by maximum conditional likelihood. Alternatively, it can simply be seen as a convex surrogate for classification error.

To be used in structured prediction, two generalizations of this optimization are needed. First, this chapter generalizes this to the case where the mapping from the input x to margins is some arbitrary set of functions $\mathcal{F}$. Then, the optimization is

$$\max_{f \in \mathcal{F}} \sum_k \left[f(x^k, y^k) - \log \sum_y \exp f(x^k, y) \right].$$

Note that this is equivalent to the previous optimization if $f(x, y) = (Wx)_y$, i.e. that $\mathcal{F}$ is the set of linear functions.

The second generalization is to add a "bias" term b^k corresponding to each datum (x^k, y^k). This is simply a term given with the dataset that biases the set of margins in a given direction. With these in place, the optimization becomes

$$\max_{f \in \mathcal{F}} \sum_k \left[f(x^k, y^k) + b^k(y^k) - \log \sum_y \exp \Big(f(x^k, y) + b^k(y) \Big) \right].$$

This loss can be solved under various function classes (albeit sometimes only to a local maximum).

1.8 Reducing Structured Learning to Logistic Regression

This section presents the main technical result of the chapter, namely that the problem of minimizing Eq. 1.18 with respect to f_α is equivalent to a logistic regression problem.

Theorem 1.4. *If f^*_α is the minimizer of Eq 1.18 for fixed messages λ, then*

$$\frac{f^*_\alpha}{\epsilon} = \operatorname*{argmax}_{f_\alpha} \sum_k \left[f_\alpha(x^k, y^k_\alpha) + b^k_\alpha(y^k_\alpha) - \log \sum_{y_\alpha} \exp \Big(f_\alpha(x^k, y_\alpha) + b^k_\alpha(y_\alpha) \Big) \right], \tag{1.19}$$

Algorithm 1.1 Structured learning via Logistic Regression

1. For all k, α, initialize $\lambda^k(y_\alpha) \leftarrow 0$.
2. Repeat until convergence:
   (a) For all k, for all α, set the bias term to

$$b_\alpha^k(y_\alpha) \leftarrow \frac{1}{\epsilon}\left(\Delta(y_\alpha^k, y_\alpha) + \sum_{\beta \subset \alpha} \lambda_\alpha^k(y_\beta) - \sum_{\gamma \supset \alpha} \lambda_\gamma^k(y_\alpha)\right).$$

   (b) For all α, solve the logistic regression problem

$$f_\alpha \leftarrow \operatorname*{argmax}_{f_\alpha \in \mathcal{F}_\alpha} \sum_{k=1}^{K}\left[f_\alpha(x^k, y_\alpha^k) + b_\alpha^k(y_\alpha^k) - \log \sum_{y_\alpha} \exp\Big(f_\alpha(x^k, y_\alpha) + b_\alpha^k(y_\alpha)\Big)\right],$$

   (c) For all k, for all α, form updated parameters as

$$\theta^k(y_\alpha) \leftarrow \epsilon f_\alpha(x^k, y_\alpha) + \Delta(y_\alpha^k, y_\alpha).$$

   (d) For all k, perform a fixed number of message-passing iterations to update λ^k using θ^k. (Eq. 1.16)

where the set of biases are defined as

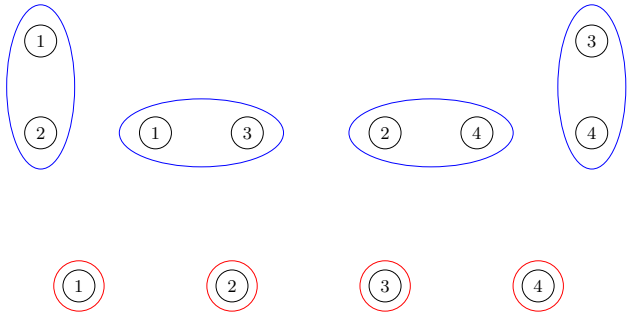
$$b_\alpha^k(y_\alpha) = \frac{1}{\epsilon}\left(\Delta(y_\alpha^k, y_\alpha) + \sum_{\beta \subset \alpha} \lambda_\alpha(y_\beta) - \sum_{\gamma \supset \alpha} \lambda_\gamma(y_\alpha)\right). \tag{1.20}$$

The proof of this theorem, which is given in the appendix, essentially consists of just substituting the value of $A(\lambda, \theta)$ from Eq. 1.15 into $\sum_k \left[-F(x^k, y^k) + A\left(\lambda^k, \theta_F^k\right)\right]$ and performing some manipulations.

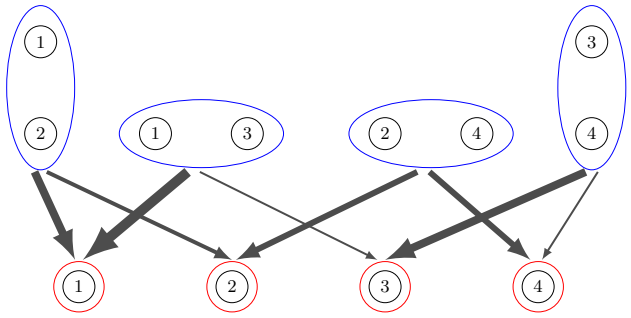
Using this result, learning can simply alternate between message-passing updates (which minimize 1.18 with respect to λ) and logistic regression updates to f_α (which minimize with respect to F). Algorithm 1.1 summarizes this approach.

1.9 Function Classes

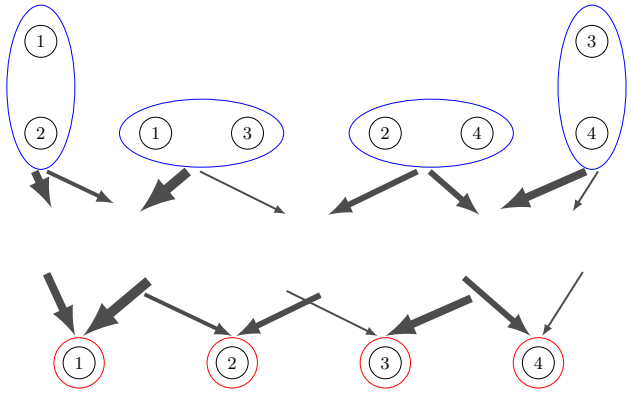
All the function classes considered in this paper assume that the input vector x has a subvector x_α corresponding to each factor α. Then the function f_α for factor α will depend on x only through x_α. Thus, through a slight abuse of notation, it is convenient to write $f_\alpha(x, y_\alpha)$ as $f_\alpha(x_\alpha, y_\alpha)$ to emphasize that it only depends on the subvector x_α.



(a) Step 1: Solve Logistic Regression Problems (similar to piecewise learning)



(b) Step 2: Update Messages



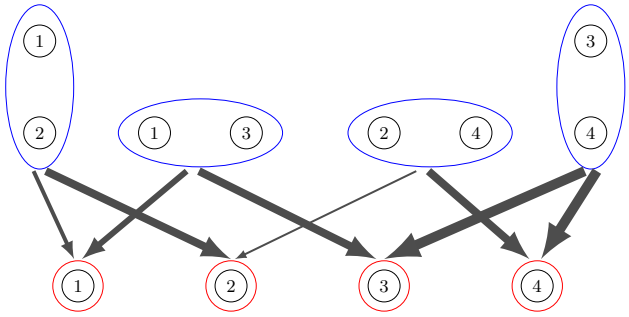
(c) Step 3: Solve biased logistic regression problems

Figure 1.3: An example of learning on the graph from Fig. 1.2

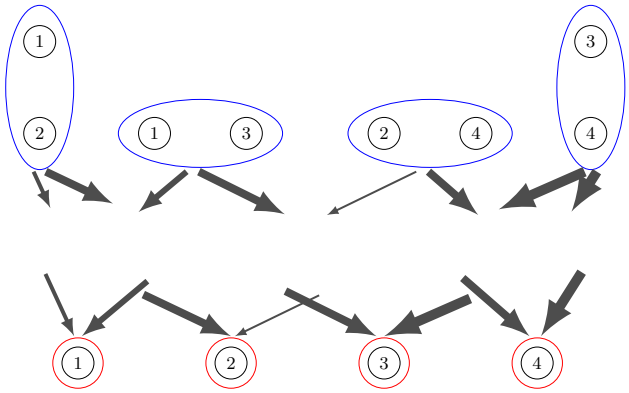
1.9.1 Zero

For comparison purposes, the experiments will sometimes use the set $\mathcal{F}_\alpha$ of functions that just consists of the "zero" function

$$f_\alpha(x_\alpha, y_\alpha) = 0.$$



(a) Step 4: Update Messages



(b) Step 5: Solve biased logistic regression problems

Figure 1.4: An example of learning (continued)

As there is only a single element f_α in $\mathcal{F}_\alpha$, optimizing a logistic loss is trivial.

1.9.2 Constant

Another simple class of functions is the set of "constant" functions

$$f_\alpha(x_\alpha, y_\alpha) = f_\alpha(y_\alpha)$$

that do not depend on x_α. These can be thought of simply as a table of values, one for each configuration y_α. Algorithmically, this class can be optimized over by fitting a linear function (as described in the following subsection) with single constant feature.

1.9.3 Linear

The most popular set of potential functions to be used in structured prediction is the linear functions. If $f_\alpha \in \mathcal{F}_\alpha$,

$$f_\alpha(x_\alpha, y_\alpha) = (W x_\alpha)_{y_\alpha}, \tag{1.21}$$

for some matrix W. Here, Eq. 1.21 should be understood as multiplying the vector x_α with the matrix W, and then selecting the component corresponding to y_α.

Optimizing a logistic loss under this function class can be easily done by gradient-based methods that optimize over W. The following experiments use limited-memory BFGS to perform this optimization.

1.9.4 Boosted Decision Trees

Trees, or ensembles of trees, are another popular choice of potential function (Shotton et al., 2009; Gould et al., 2008; Xiao and Quan, 2009; Ladicky et al., 2009; Winn and Shotton, 2006; Nowozin et al., 2011; Schroff et al., 2008). Here, these are learned following the basic strategy of gradient boosting (Friedman, 1999). The basic idea is to repeatedly induce decision trees to reduce the logistic loss

$$L_k(f_\alpha) = f_\alpha(x_\alpha^k, y_\alpha^k) + b_\alpha^k(y_\alpha^k) - \log \sum_{y_\alpha} \exp\Big(f_\alpha(x_\alpha^k, y_\alpha) + b_\alpha^k(y_\alpha)\Big).$$

This is done by initializing $f_\alpha(x_\alpha, y_\alpha) = 0$ and then repeating the following steps:

1. For each datum calculate the gradient of the loss $g^k(y_\alpha) = dL_k / df_\alpha(x_\alpha^k, y_\alpha^k)$
2. Induce a regression tree t to minimize $\sum_k \sum_{y_\alpha} \left(g^k(y_\alpha) - t(x_\alpha^k, y_\alpha^k)\right)^2$.
3. Leaving the split points of t fixed, adjust the values of the leaf nodes (via L-BFGS) to minimize the empirical risk $\sum_k L_k(f_\alpha + t)$.
4. Multiply t by a step length ν and add it to the ensemble as

$$f_\alpha(x_\alpha, y_\alpha) \leftarrow f_\alpha(x_\alpha, y_\alpha) + \nu \times t(x_\alpha, y_\alpha).$$

Several details are needed to fully describe the method:

First, note the a single regression tree t is induced for all classes, rather than inducing one tree for each class, as is more common. To do this, a tree needs to be induced to minimize a multivariate squared loss. This can be written as finding t to minimize $\sum_k ||g^k - t(x_\alpha^k)||^2$ where g^k and $t(x_\alpha^k)$ are vectors of the values $g^k(y_\alpha)$ and $t(x_\alpha^k, y_\alpha)$ respectively. As is typical when

fitting regression trees, this is done greedily: the algorithm repeatedly picks a dimension and split point to divide the data into two groups, such that the sum of squared distances of all points to the mean of g^k in the corresponding group is minimized. Implemented naively, this would require on the order of $D_{\text{in}}K^2D_{\text{out}}$ operations[2], where K is the number of data and D_{in} is the number of input dimensions, and D_{out} is the number out output dimensions. However, exploiting a recursion in the structure of the costs can reduce this to the order of $D_{\text{in}}K(\log K + D_{\text{out}})$ operations[3].

Second, when inducing the tree, splits are only considered that leave at least 1% of the original data in each leaf node. Nodes are not split at all if they contain less than 2.5% of the original data.

Finally, two heuristics are borrowed from stochastic gradient boosting. When inducing a tree in step 2 above, and also when adjusting the values of the leaf nodes in step 3, only a random subset of 10,000 elements of the data are used. (The same subset for both sets, but randomly selected for each iteration.) This greatly speeds up computation, and induces some randomness into the selected trees. Finally, a step size of $\nu = .25$ is used, which compensates for the randomness and improves test-set performance. A total of 200 boosting iterations are performed.

1.9.5 Multi-Layer Perceptrons

Multi-Layer perceptrons are also sometimes used as potential functions (He et al., 2004; Silberman and Fergus, 2011). These experiments use a simple multi-layer perceptron with a single hidden layer, which can be written as

$$f_\alpha(x_\alpha, y_\alpha) = (W\sigma(Vx_\alpha))_{y_\alpha}.$$

This can be understood as multiplying the input x_α by a matrix V and passing it through the "sigmoid" function σ (that applies a tanh elementwise) to obtain a hidden representation $\sigma(Vx_\alpha)$. W maps this hidden representation to the ouput space. In all experiments, the hidden representation has 100 elements. To fit a logistic loss, stochastic gradient descent is used with mini-batches of size 100, and a step size of 0.25 for factors corresponding to single variables, and 0.05 for pairs. A momentum constant of .9 is used, meaning each step taken is a combination of .9 of the last step and .1 of the current gradient.

---

2. There are K unique split points in each of the D_{in} dimensions, and checking the cost of each can be done with KD_{out} operations.
3. Here, $D_{\text{in}}K\log K$ is the cost of sorting each input dimension, and after sorting, all split points in each single input dimension can be evaluated in $K\text{D}_{\text{out}}$ time.

Of course, gradient methods will only find a local optimum of the logistic loss. To at least guarantee improvement in each iteration, parameters are initialized to the those from the previous iteration.

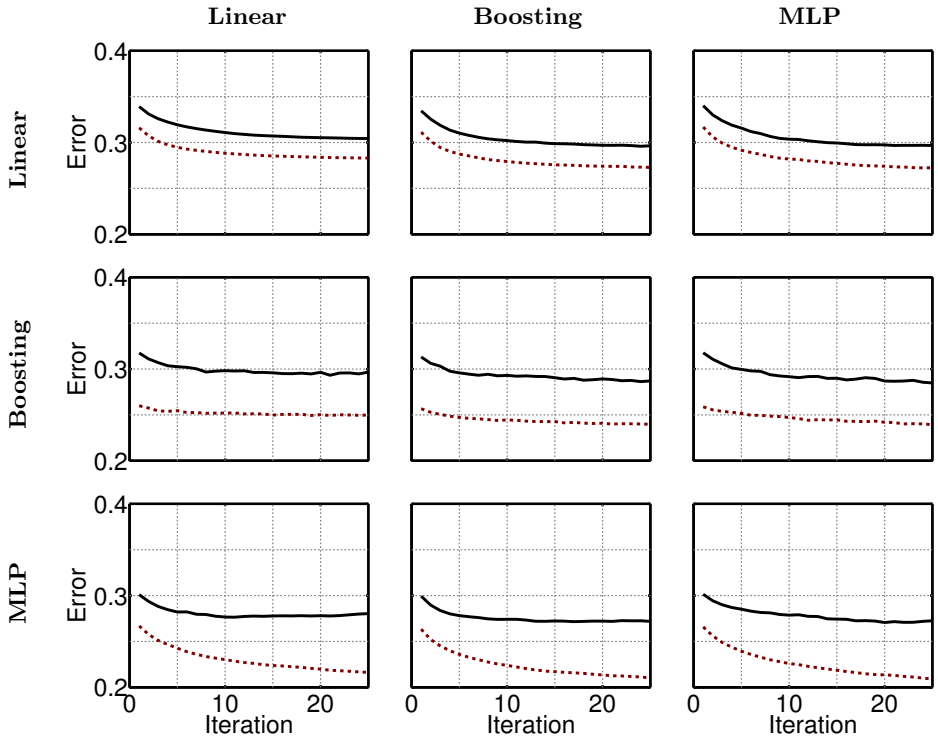


Figure 1.5: Training (dashed) and test (solid) error rates as a function of the number of learning iterations for each combination of univariate (rows) and pairwise (columns) potential functions.

1.10 Example

This section presents a simple example of learning using the Stanford Background dataset (Gould et al., 2009), split into a training set of 572 and a test set of 143 images, each of resolution roughly 320 x 240. For each pixel, 41 univariate features were computed, including the RGB value, the x and y horizontal position, normalized to [0,1], and a 36 component Histogram of Oriented Gradients (Dalal and Triggs, 2005). There are 3 pairwise features, consisting of a constant of 1, the l_2 distance of the RGB components of the two pixels, and the output of a Sobel edge filter. Images were reduced to

	Zero	Constant	Linear	Boosting	MLP
Zero	.853 / .863	.641 / .673	.553 / .593	.470 / .483	.497 / .518
Constant	.769 / .793	.640 / .673	.553 / .593	.470 / .485	.497 / .518
Linear	.322 / .345	.304 / .329	.283 / .304	.272 / .296	.272 / .296
Boosting	.289 / .331	.259 / .304	.249 / .296	.239 / .287	.239 / .284
MLP	.262 / .310	.226 / .281	.216 / .280	.210 / .271	.209 / .272

Table 1.1: Train / Test Error rates for each combination of univariate (rows) and pairwise (columns) potential functions.

20% resolution before computing features.

Learning proceeds through a set of iterations. In each learning iteration, the univariate potential function is updated, followed by 25 message-passing iterations (each of which passes over the entire image from top-left to bottom right, then in the reverse order). This is followed by an update to the pairwise potentials, and another 25 message-passing iterations. There were 25 total learning iterations.

The univariate training and test error rates are shown in Fig. 1.5 and Table 1.1. It is easy to see that more powerful potential functions lead to lower training errors, but this is somewhat offset by more overfitting.

1.11 Conclusions

This chapter builds on two streams of previous work for structured learning. In the first, the likelihood is replaced with a piecewise (Sutton and Mccallum, 2009) or pseudolikelihood approximation. This decomposes the learning objective into a sum of local objectives. These can be optimized efficiently, and make it possible to use nonlinear function classes, such as trees or multi-layer perceptrons. The downside of these training methods is that the approximation to the likelihood can sometimes be weak, leading to poor performance of the learned model in the face of joint prediction.

The second stream of related research is based on phrasing structured learning with linear predictors as a joint optimization of inference messages and model parameters (Hazan and Urtasun, 2012; Meshi et al., 2010), alternating between optimization of each. These optimize a loss that deals with approximate inference in a principled way, but are fairly specific to linear energies.

The main result here is that, when pursing this latter strategy, optimizing model parameters with fixed inference messages leads to a set of logistic regression problems, each biased by the current messages. This yields an al-

gorithm alternating between message-passing updates and logistic regression problems. Given the high degree of similarity between logistic regression and piecewise training, one can view this as using message-passing to iteratively tighten a piecewise-style learning objective towards better joint prediction. Additionally, it is easy to use any function class over which a logistic loss can be fit, such as ensembles of trees or multi-layer perceptrons.

Future work should understand the convergence rates of a procedure that alternates between message-passing an logistic regression updates. Given the existing results on convergence rates for this style of message-passing inference (Meshi et al., 2012) and standard results for convex optimization, this could lead to joint convergence rates.

Another possible extension of this procedure would be to consider more general entropy approximations. That is, one might extend the approach described here to the case where the entropy smoothing takes the form of $\sum_\alpha \epsilon_\alpha H(\mu_\alpha)$, with different entropy weights ϵ_α for different regions α. This would allow the use of this style of algorithm for so-called "surrogate likelihood" (Wainwright, 2006) training, in which the likelihood is approximated using an algorithm like loopy belief propagation. Now, if $\epsilon_\alpha > 0$ for all α, this extension is trivial. However, standard entropy approximations involve the use of negative weights, where $\epsilon_\alpha < 0$ for some α. These terms introduce non-convexity, which defeats obvious extensions of the method described here.

Appendix: Proofs

This appendix contains proofs of all the main results of the paper.

Boundedness of Entropy Smoothing

Proof of Theorem 1.1. We can write

$$\begin{aligned} l(x,y;F) - l_1(x,y;F) &= -F(x,y) + \max_{\mu\in\mathcal{M}} \left(\theta\cdot\mu + \sum_\alpha \epsilon H(\mu_\alpha)\right) \\ &\quad + F(x,y) - \max_{\mu\in\mathcal{M}} \theta\cdot\mu \\ &= \max_{\mu\in\mathcal{M}} \left(\theta\cdot\mu + \sum_\alpha \epsilon H(\mu_\alpha)\right) - \max_{\mu\in\mathcal{M}} \theta\cdot\mu \\ &= \theta\cdot\mu' - \theta\cdot\mu^* + \sum_\alpha \epsilon H(\mu'_\alpha) \\ &\leq \epsilon \sum_\alpha \log|y_\alpha|. \end{aligned}$$

where we have defined $\mu^* = \text{argmax}_{\mu\in\mathcal{M}}\, \theta\cdot\mu$ and $\mu' = \text{argmax}_{\mu\in\mathcal{M}}\, \theta\cdot\mu + \epsilon\sum_\alpha H(\mu_\alpha)$. The last line follows from the fact that $\theta\cdot\mu^* \geq \theta\cdot\mu'$, and that $H(\mu'_\alpha) \leq \log|y_\alpha|$.

□

Dual Representation of A

This section presents a proof which requires the following standard result.

Lemma 1.5. *The conjugate of the entropy is the "log-sum-exp" function. Formally,*

$$\max_{p:\sum_i p_i=1, p_i\geq 0} \theta\cdot p - \epsilon\sum_i p_i\log p_i = \epsilon\log\sum_i \exp\frac{\theta_i}{\epsilon}.$$

Moreover, the maximizing p is given by

$$p = \frac{\exp(\theta/\epsilon)}{\sum_i \exp(\theta_i/\epsilon)}$$

Proof of Theorem 1.2. Firstly, we transform the optimization of A into the following form, where we use a set $\mathcal{N}$ to denote the set of locally normalized distributions, but explicitly enforce marginalization constraints.

$$A(\theta) = \max_{\mu\in\mathcal{N}} \theta\cdot\mu + \epsilon\sum_\alpha H(\mu_\alpha) \tag{1.22}$$

$$\text{s.t.} \quad \mu_{\alpha\beta}(y_\beta) = \mu_\beta(y_\beta)\,\forall\beta\subset\alpha, y_\beta \tag{1.23}$$

Next, if one introduces a set of Lagrange multipliers

$$\lambda = \{\lambda_\alpha(y_\beta), \forall \beta \subset \alpha, y_\alpha\},$$

one can write A in the form

$$\begin{aligned} A(\theta) &= \max_{\mu \in \mathcal{N}} \min_{\lambda} \theta \cdot \mu + \epsilon \sum_\alpha H(\mu_\alpha) && (1.24)\\ &\quad + \sum_\alpha \sum_{\beta \subset \alpha} \lambda_\alpha(x_\beta)(\mu_{\alpha\beta}(y_\beta) - \mu_\beta(y_\beta)). && (1.25) \end{aligned}$$

By Sion's theorem (Sion, 1958), it is possible to interchange the maximum and minimum. Thus, if we define

$$\begin{aligned} A(\lambda, \theta) &= \max_{\mu \in \mathcal{N}} \theta \cdot \mu + \epsilon \sum_\alpha H(\mu_\alpha) && (1.26)\\ &\quad + \sum_\alpha \sum_{\beta \subset \alpha} \lambda_\alpha(x_\beta)(\mu_{\alpha\beta}(y_\beta) - \mu_\beta(y_\beta)), && (1.27) \end{aligned}$$

we can represent $A(\theta)$ simply as $A(\theta) = \min_\lambda A(\lambda, \theta)$.

Now, we can re-write this objective as

$$\sum_\alpha \left(\sum_{y_\alpha} \theta(y_\alpha)\mu(y_\alpha) + \sum_\alpha \sum_{\beta \subset \alpha} \lambda_\alpha(x_\beta)(\mu_{\alpha\beta}(y_\beta) - \mu_\beta(y_\beta)) \right) + \epsilon \sum_\alpha H(\mu_\alpha) \quad (1.28)$$

$$= \sum_\alpha \left(\sum_{y_\alpha} \left(\theta(y_\alpha) + \sum_{\beta \subset \alpha} \lambda_\alpha(x_\beta) - \sum_{\gamma \supset \alpha} \lambda_\gamma(x_\alpha) \right) \mu_\alpha(y_\alpha) + \epsilon H(\mu_\alpha) \right). \quad (1.29)$$

It remains to actually calculate the μ that maximizes Eq. 1.27. The key observation here is that the variables $\mu(y_\alpha)$ corresponding to each factor can be optimized independently. If we consider an arbitrary factor α, the problem is to maximize

$$\begin{aligned} \max_{\mu_\alpha} \quad & \sum_{y_\alpha} \theta(y_\alpha)\mu_\alpha(y_\alpha) + \epsilon H(\mu_\alpha) && (1.30)\\ & + \sum_{\beta \subset \alpha} \lambda_\alpha(x_\beta)\mu_{\alpha\beta}(y_\beta) - \sum_{\gamma \supset \alpha} \lambda_\gamma(x_\alpha)\mu_\alpha(y_\alpha), && (1.31)\\ \text{s.t.} \quad & \sum_{y_\alpha} \mu_\alpha(y_\alpha) = 1 && (1.32)\\ & \mu_\alpha(y_\alpha) \geq 0 && (1.33) \end{aligned}$$

We can re-write this as

$$\max_{\mu_\alpha} \sum_{y_\alpha} \left(\theta(y_\alpha) + \sum_{\beta \subset \alpha} \lambda_\alpha(x_\beta) - \sum_{\gamma \supset \alpha} \lambda_\gamma(x_\alpha) \right) \mu_\alpha(y_\alpha) + \epsilon H(\mu_\alpha) \quad (1.34)$$

$$\text{s.t.} \sum_{y_\alpha} \mu_\alpha(y_\alpha) = 1 \quad (1.35)$$

$$\mu_\alpha(y_\alpha) \geq 0 \quad (1.36)$$

Using the above lemma, we see that the solution is

$$\mu_\alpha(y_\alpha) \propto \exp\left(\frac{1}{\epsilon} \left(\theta(y_\alpha) + \sum_{\beta \subset \alpha} \lambda_\alpha(x_\beta) - \sum_{\gamma \supset \alpha} \lambda_\gamma(x_\alpha) \right) \right), \quad (1.37)$$

with a corresponding function value of

$$\epsilon \log \sum_{y_\alpha} \exp\left(\frac{1}{\epsilon} \left(\theta(y_\alpha) + \sum_{\beta \subset \alpha} \lambda_\alpha(x_\beta) - \sum_{\gamma \supset \alpha} \lambda_\gamma(x_\alpha) \right) \right). \quad (1.38)$$

□

Message-Passing Update Equations

Proof of Theorem 1.3. It is not hard to see that, after update, the new marginals will obey the conditions

$$\mu'_\nu(y_\nu) \propto \mu_\nu(y_\nu) \exp\left(-\frac{1}{\epsilon} \sum_{\eta \supset \nu} \delta_\eta(y_\nu) \right) \quad (1.39)$$

$$\mu'_\eta(y_\eta) \propto \mu_\eta(y_\eta) \exp\left(\frac{1}{\epsilon} \delta_\eta(y_\nu) \right). \quad (1.40)$$

$$\mu'_{\eta\nu}(y_\nu) \propto \mu_{\eta\nu}(y_\nu) \exp\left(\frac{1}{\epsilon} \delta_\eta(y_\nu) \right). \quad (1.41)$$

Now, if the marginals are updated as above then, for all $\eta \supset \nu$

$$\mu'_{\eta\nu}(y_\nu) \propto \mu_{\eta\nu}(y_\eta) \exp\left(\frac{\log \mu_\nu(y_\nu) + \sum_{\eta' \supset \nu} \log \mu_{\eta'\nu}(y_\nu)}{1 + N_\nu} - \log \mu_{\eta\nu}(y_\nu)\right) \tag{1.42}$$

$$= \exp\left(\log \mu_\nu(y_\nu) + \sum_{\eta' \supset \nu} \log \mu_{\eta'\nu}(y_\nu)\right)^{1/(1+N_\nu)}, \tag{1.43}$$

which can be seen to be equal to Eq. 1.17. Similarly, for ν itself,

$$\mu'_\nu(y_\nu) \propto \mu_\nu(y_\nu) \exp\left(-\frac{1}{\epsilon} \sum_{\eta \supset \nu} \delta_\eta(y_\nu)\right) \tag{1.44}$$

$$= \mu_\nu(y_\nu) \exp\Big(-\frac{1}{1+N_\nu} \sum_{\eta \supset \nu} \left(\log \mu_\nu(y_\nu) + \sum_{\eta' \supset \nu} \log \mu_{\eta'\nu}(y_\nu)\right) + \sum_{\eta \supset \nu} \log \mu_\eta(y_\nu)\Big) \tag{1.45}$$

$$= \mu_\nu(y_\nu) \exp\big(-\frac{N_\nu}{1+N_\nu} \left(\log \mu_\nu(y_\nu) + \sum_{\eta' \supset \nu} \log \mu_{\eta'\nu}(y_\nu)\right) + \sum_{\eta \supset \nu} \log \mu_\eta(y_\nu)\big) \tag{1.46}$$

$$= \mu_\nu(y_\nu) \mu_\nu(y_\nu)^{-N_\nu/(1+N_\nu)} \prod_{\eta' \supset \nu} \mu_{\eta'\nu}(y_\nu)^{-N_\nu/(1+N_\nu)} \prod_{\eta \supset \nu} \mu_{\eta\nu}(y_\nu), \tag{1.47}$$

which again is equal to Eq. 1.17.

□

Logistic Regression Reduction

Proof of Theorem 1.4. If we consider minimizing $\sum_k \left[-F(x^k, y^k) + A\left(\lambda^k, \theta_F^k\right)\right]$ with respect to a single f_α, it is easy to see from Eqs. 1.2 and 1.15 that the problem is

$$\operatorname*{argmin}_{f_\alpha \in \mathcal{F}_\alpha} \sum_k \left[-f_\alpha(x^k, y^k) + \epsilon \log \sum_{y_\alpha} \exp\left(\frac{1}{\epsilon}\left(\theta_F^k(y_\alpha) + \sum_{\beta \subset \alpha} \lambda_\alpha(y_\beta) - \sum_{\gamma \supset \alpha} \lambda_\gamma(y_\alpha)\right)\right)\right],$$

substituting the definition of θ_F^k as $\theta_F^k(y_\alpha) = f_\alpha(x^k, y_\alpha) + \Delta_\alpha(y_\alpha^k, y_\alpha)$, and using the above set of biases, this is

$$\operatorname*{argmin}_{f_\alpha \in \mathcal{F}_\alpha} \sum_k \left[-f_\alpha(x^k, y^k) + \epsilon \log \sum_{y_\alpha} \exp\left(\frac{1}{\epsilon} f_\alpha(x^k, y_\alpha) + b_\alpha^k(y_\alpha) \right) \right].$$

As adding or multiplying by a constant does not affect the minimizer, this is

$$\operatorname*{argmin}_{f_\alpha \in \mathcal{F}_\alpha} \sum_k \left[-\frac{1}{\epsilon} f_\alpha(x^k, y^k) - b_\alpha(x^k, y^k) + \log \sum_{y_\alpha} \exp\left(\frac{1}{\epsilon} f_\alpha(x^k, y_\alpha) + b_\alpha^k(y_\alpha) \right) \right].$$

Finally, observing that $\operatorname{argmin} g(\frac{1}{\epsilon}\cdot) = \epsilon \operatorname{argmin} g(\cdot)$, and that $\operatorname{argmin} -g(\cdot) = \operatorname{argmax} g(\cdot)$ gives the result.

□

1.13 References

K. Crammer and Y. Singer. On the algorithmic implementation of multiclass kernel-based vector machines. *Journal of Machine Learning Research*, 2:265–292, 2002.

N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, 2005.

T. Finley and T. Joachims. Training structural SVMs when exact inference is intractable. In *Proceedings of the International Conference on Machine Learning*, 2008.

J. H. Friedman. Stochastic gradient boosting. *Computational Statistics and Data Analysis*, 38:367–378, 1999.

S. Gould, J. Rodgers, D. Cohen, G. Elidan, and D. Koller. Multi-class segmentation with relative location prior. *International Journal of Computer Vision*, 80(3): 300–316, 2008.

S. Gould, R. Fulton, and D. Koller. Decomposing a scene into geometric and semantically consistent regions. In *Proceeding of International Conference on Computer Vision*, 2009.

T. Hazan and R. Urtasun. Efficient learning of structured predictors in general graphical models. *CoRR*, abs/1210.2346, 2012.

X. He, R. S. Zemel, and M. Á. Carreira-Perpiñán. Multiscale conditional random fields for image labeling. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, 2004.

T. Heskes. Convexity arguments for efficient minimization of the Bethe and Kikuchi free energies. *Journal of Artificial Intelligence Research*, 26:153–190, 2006.

D. Koller and N. Friedman. *Probabilistic Graphical Models: Principles and Techniques*. MIT Press, 2009.

L. Ladicky, C. Russell, P. Kohli, and P. H. S. Torr. Associative hierarchical CRFs for object class image segmentation. In *Proceedings of International Conference on Computer Vision*, 2009.

O. Meshi, D. Sontag, T. Jaakkola, and A. Globerson. Learning efficiently with approximate inference via dual losses. In *Proceedings of the International Conference on Machine Learning*, 2010.

O. Meshi, T. Jaakkola, and A. Globerson. Convergence rate analysis of MAP coordinate minimization algorithms. In *Proceedings of the Conference on Neural Information Processing Systems*. 2012.

A. Nedi and A. Ozdaglar. Subgradient methods for saddle-point problems. *Journal of Optimization Theory and Applications*, pages 205–228, 2009.

S. Nowozin, C. Rother, S. Bagon, T. Sharp, B. Yao, and P. Kohli. Decision tree fields. In *Proceedings of International Conference on Computer Vision*, 2011.

N. Ratliff, J. A. D. Bagnell, and M. Zinkevich. (Online) subgradient methods for structured prediction. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, 2007.

F. Schroff, A. Criminisi, and A. Zisserman. Object class segmentation using random forests. In *Proceedings of the British Machine Vision Conference*, 2008.

J. Shotton, J. M. Winn, C. Rother, and A. Criminisi. Textonboost for image understanding: Multi-class object recognition and segmentation by jointly modeling texture, layout, and context. *International Journal of Computer Vision*, 81(1): 2–23, 2009.

N. Silberman and R. Fergus. Indoor scene segmentation using a structured light sensor. In *Proceedings of International Conference on Computer Vision Workshops*, 2011.

M. Sion. On general minimax theorems. *Pacific Journal of Mathematics*, 8(1): 171–176, 1958.

D. Sontag and T. Jaakkola. Tree block coordinate descent for MAP in graphical models. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, 2009.

C. Sutton and A. Mccallum. Piecewise training for structured prediction. *Machine Learning*, 77:165–194, 2009.

B. Taskar, C. Guestrin, and D. Koller. Max-margin markov networks. In *Proceedings of the Conference on Neural Information Processing Systems*, 2003.

I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun. Large margin methods for structured and interdependent output variables. *J. Mach. Learn. Res.*, 6: 1453–1484, 2005.

V. V. Vazirani. *Approximation algorithms*. Springer-Verlag, 2001.

M. Wainwright and M. Jordan. Graphical models, exponential families, and variational inference. *Foundations and Trends in Machine Learning*, 1(1-2):1–305, 2008.

M. J. Wainwright. Estimating the "wrong" graphical model: benefits in the computation-limited setting. *Journal of Machine Learning Research*, 7:1829–1859, 2006.

J. M. Winn and J. Shotton. The layout consistent random field for recognizing and segmenting partially occluded objects. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, 2006.

J. Xiao and L. Quan. Multiple view semantic segmentation for street view images. In *Proceedings of International Conference on Computer Vision*, 2009.