MLE has a closed form solution (Lauritzen 1996). In particular, for tabular potentials we have

$$\hat{\psi}_c(\mathbf{y}_c = k) = \frac{\sum_{i=1}^N \mathbb{I}(\mathbf{y}_{i,c} = k)}{N} \tag{19.61}$$

and for Gaussian potentials, we have

$$\hat{\boldsymbol{\mu}}_c = \frac{\sum_{i=1}^N \mathbf{y}_{ic}}{N}, \; \hat{\boldsymbol{\Sigma}}_c = \frac{\sum_i (\mathbf{y}_{ic} - \hat{\boldsymbol{\mu}}_c)(\mathbf{x}_{ic} - \hat{\boldsymbol{\mu}}_c)^T}{N} \tag{19.62}$$

By using conjugate priors, we can also easily compute the full posterior over the model parameters in the decomposable case, just as we did in the DGM case. See (Lauritzen 1996) for details.

19.6 Conditional random fields (CRFs)

A **conditional random field** or **CRF** (Lafferty et al. 2001), sometimes a **discriminative random field** (Kumar and Hebert 2003), is just a version of an MRF where all the clique potentials are conditioned on input features:

$$p(\mathbf{y}|\mathbf{x}, \mathbf{w}) = \frac{1}{Z(\mathbf{x}, \mathbf{w})} \prod_c \psi_c(\mathbf{y}_c|\mathbf{x}, \mathbf{w}) \tag{19.63}$$

A CRF can be thought of as a **structured output** extension of logistic regression. We will usually assume a log-linear representation of the potentials:

$$\psi_c(\mathbf{y}_c|\mathbf{x}, \mathbf{w}) = \exp(\mathbf{w}_c^T \boldsymbol{\phi}(\mathbf{x}, \mathbf{y}_c)) \tag{19.64}$$

where $\boldsymbol{\phi}(\mathbf{x}, \mathbf{y}_c)$ is a feature vector derived from the global inputs $\mathbf{x}$ and the local set of labels $\mathbf{y}_c$. We will give some examples below which will make this notation clearer.

The advantage of a CRF over an MRF is analogous to the advantage of a discriminative classifier over a generative classifier (see Section 8.6), namely, we don't need to "waste resources" modeling things that we always observe. Instead we can focus our attention on modeling what we care about, namely the distribution of labels given the data.

Another important advantage of CRFs is that we can make the potentials (or factors) of the model be data-dependent. For example, in image processing applications, we may "turn off" the label smoothing between two neighboring nodes s and t if there is an observed discontinuity in the image intensity between pixels s and t. Similarly, in natural language processing problems, we can make the latent labels depend on global properties of the sentence, such as which language it is written in. It is hard to incorporate global features into generative models.

The disadvantage of CRFs over MRFs is that they require labeled training data, and they are slower to train, as we explain in Section 19.6.3. This is analogous to the strengths and weaknesses of logistic regression vs naive Bayes, discussed in Section 8.6.

19.6.1 Chain-structured CRFs, MEMMs and the label-bias problem

The most widely used kind of CRF uses a chain-structured graph to model correlation amongst neighboring labels. Such models are useful for a variety of sequence labeling tasks (see Section 19.6.2).

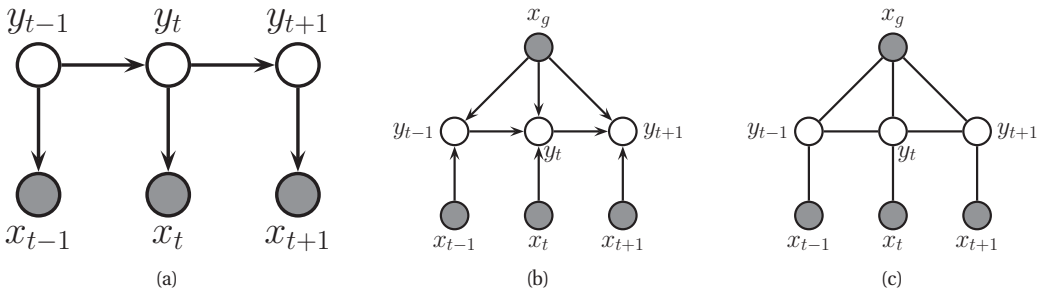


Figure 19.14 Various models for sequential data. (a) A generative directed HMM. (b) A discriminative directed MEMM. (c) A discriminative undirected CRF.

Traditionally, HMMs (discussed in detail in Chapter 17) have been used for such tasks. These are joint density models of the form

$$p(\mathbf{x}, \mathbf{y}|\mathbf{w}) = \prod_{t=1}^{T} p(y_t|y_{t-1}, \mathbf{w}) p(\mathbf{x}_t|y_t, \mathbf{w}) \tag{19.65}$$

where we have dropped the initial $p(y_1)$ term for simplicity. See Figure 19.14(a). If we observe both $\mathbf{x}_t$ and y_t for all t, it is very easy to train such models, using techniques described in Section 17.5.1.

An HMM requires specifying a generative observation model, $p(\mathbf{x}_t|y_t, \mathbf{w})$, which can be difficult. Furthemore, each $\mathbf{x}_t$ is required to be local, since it is hard to define a generative model for the whole stream of observations, $\mathbf{x} = \mathbf{x}_{1:T}$.

An obvious way to make a discriminative version of an HMM is to "reverse the arrows" from y_t to $\mathbf{x}_t$, as in Figure 19.14(b). This defines a directed discriminative model of the form

$$p(\mathbf{y}|\mathbf{x}, \mathbf{w}) = \prod_{t} p(y_t|y_{t-1}, \mathbf{x}, \mathbf{w}) \tag{19.66}$$

where $\mathbf{x} = (\mathbf{x}_{1:T}, \mathbf{x}_g)$, $\mathbf{x}_g$ are global features, and $\mathbf{x}_t$ are features specific to node t. (This partition into local and global is not necessary, but helps when comparing to HMMs.) This is called a **maximum entropy Markov model** or **MEMM** (McCallum et al. 2000; Kakade et al. 2002).

An MEMM is simply a Markov chain in which the state transition probabilities are conditioned on the input features. (It is therefore a special case of an input-output HMM, discussed in Section 17.6.3.) This seems like the natural generalization of logistic regression to the structured-output setting, but it suffers from a subtle problem known (rather obscurely) as the **label bias** problem (Lafferty et al. 2001). The problem is that local features at time t do not influence states prior to time t. This follows by examining the DAG, which shows that $\mathbf{x}_t$ is d-separated from y_{t-1} (and all earlier time points) by the v-structure at y_t, which is a hidden child, thus blocking the information flow.

To understand what this means in practice, consider the part of speech (POS) tagging task. Suppose we see the word "banks"; this could be a verb (as in "he banks at BoA"), or a noun (as in "the river banks were overflowing"). Locally the POS tag for the word is ambiguous. However,

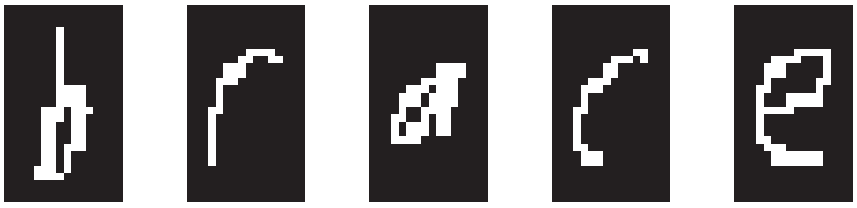


Figure 19.15 Example of handwritten letter recognition. In the word 'brace', the 'r' and the 'c' look very similar, but can be disambiguated using context. Source: (Taskar et al. 2003) . Used with kind permission of Ben Taskar.

suppose that later in the sentence, we see the word "fishing"; this gives us enough context to infer that the sense of "banks" is "river banks". However, in an MEMM (unlike in an HMM and CRF), the "fishing" evidence will not flow backwards, so we will not be able to disambiguate "banks".

Now consider a chain-structured CRF. This model has the form

$$p(\mathbf{y}|\mathbf{x},\mathbf{w}) = \frac{1}{Z(\mathbf{x},\mathbf{w})} \prod_{t=1}^{T} \psi(y_t|\mathbf{x},\mathbf{w}) \prod_{t=1}^{T-1} \psi(y_t, y_{t+1}|\mathbf{x},\mathbf{w}) \tag{19.67}$$

From the graph in Figure 19.14(c), we see that the label bias problem no longer exists, since y_t does not block the information from $\mathbf{x}_t$ from reaching other $y_{t'}$ nodes.

The label bias problem in MEMMs occurs because directed models are **locally normalized**, meaning each CPD sums to 1. By contrast, MRFs and CRFs are **globally normalized**, which means that local factors do not need to sum to 1, since the partition function Z, which sums over all joint configurations, will ensure the model defines a valid distribution. However, this solution comes at a price: we do not get a valid probability distribution over $\mathbf{y}$ until we have seen the whole sentence, since only then can we normalize over all configurations. Consequently, CRFs are not as useful as DGMs (whether discriminative or generative) for online or real-time inference. Furthermore, the fact that Z depends on all the nodes, and hence all their parameters, makes CRFs much slower to train than DGMs, as we will see in Section 19.6.3.

19.6.2 Applications of CRFs

CRFs have been applied to many interesting problems; we give a representative sample below. These applications illustrate several useful modeling tricks, and will also provide motivation for some of the inference techniques we will discuss in Chapter 20.

19.6.2.1 Handwriting recognition

A natural application of CRFs is to classify hand-written digit strings, as illustrated in Figure 19.15. The key observation is that locally a letter may be ambiguous, but by depending on the (unknown) labels of one's neighbors, it is possible to use context to reduce the error rate. Note that the node potential, $\psi_t(y_t|\mathbf{x}_t)$, is often taken to be a probabilistic discriminative classifier,

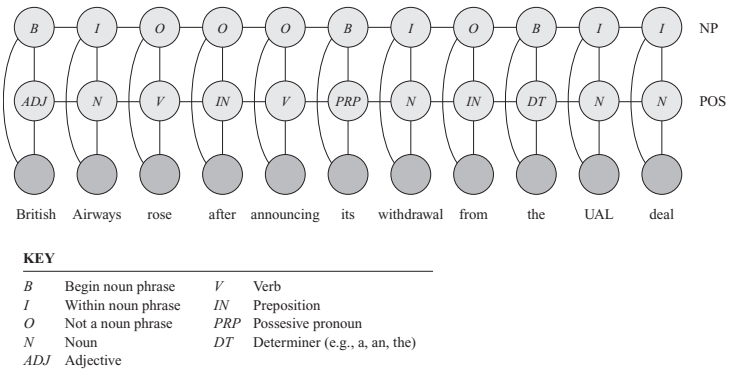


Figure 19.16 A CRF for joint POS tagging and NP segmentation. Source: Figure 4.E.1 of (Koller and Friedman 2009). Used with kind permission of Daphne Koller.

such as a neural network or RVM, that is trained on isolated letters, and the edge potentials, $\psi_{st}(y_s, y_t)$, are often taken to be a language bigram model. Later we will discuss how to train all the potentials jointly.

19.6.2.2 Noun phrase chunking

One common NLP task is **noun phrase chunking**, which refers to the task of segmenting a sentence into its distinct noun phrases (NPs). This is a simple example of a technique known as **shallow parsing**.

In more detail, we tag each word in the sentence with B (meaning beginning of a new NP), I (meaning inside a NP), or O (meaning outside an NP). This is called **BIO** notation. For example, in the following sentence, the NPs are marked with brackets:

```
   B      I        O     O      O         B    I         O    B   I   I
(British Airways) rose after announcing (its withdrawl) from (the UAI deal)
```

(We need the B symbol so that we can distinguish `I I`, meaning two words within a single NP, from `B B`, meaning two separate NPs.)

A standard approach to this problem would first convert the string of words into a string of POS tags, and then convert the POS tags to a string of BIOs. However, such a **pipeline** method can propagate errors. A more robust approach is to build a joint probabilistic model of the form $p(\text{NP}_{1:T}, \text{POS}_{1:T}|\text{words}_{1:T})$. One way to do this is to use the CRF in Figure 19.16. The connections between adjacent labels encode the probability of transitioning between the B, I and O states, and can enforce constraints such as the fact that B must preceed I. The features are usually hand engineered and include things like: does this word begin with a capital letter, is this word followed by a full stop, is this word a noun, etc. Typically there are $\sim 1,000 - 10,000$ features per node.

The number of features has minimal impact on the inference time, since the features are observed and do not need to be summed over. (There is a small increase in the cost of

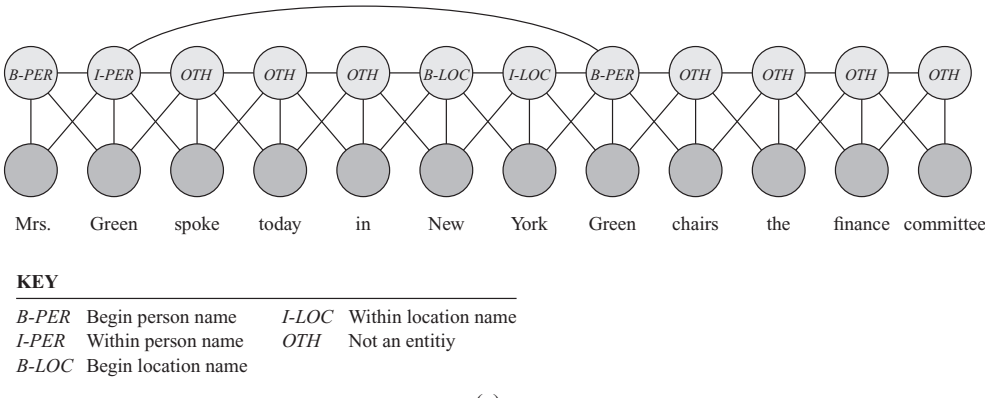


Figure 19.17 A skip-chain CRF for named entity recognition. Source: Figure 4.E.1 of (Koller and Friedman 2009). Used with kind permission of Daphne Koller.

evaluating potential functions with many features, but this is usually negligible; if not, one can use ℓ_1 regularization to prune out irrelevant features.) However, the graph structure can have a dramatic effect on inference time. The model in Figure 19.16 is tractable, since it is essentially a "fat chain", so we can use the forwards-backwards algorithm (Section 17.4.3) for exact inference in $O(T|\text{POS}|^2|\text{NP}|^2)$ time, where $|\text{POS}|$ is the number of POS tags, and $|\text{NP}|$ is the number of NP tags. However, the seemingly similar graph in Figure 19.17, to be explained below, is computationally intractable.

19.6.2.3 Named entity recognition

A task that is related to NP chunking is **named entity extraction**. Instead of just segmenting out noun phrases, we can segment out phrases to do with people and locations. Similar techniques are used to automatically populate your calendar from your email messages; this is called **information extraction**.

A simple approach to this is to use a chain-structured CRF, but to expand the state space from BIO to B-Per, I-Per, B-Loc, I-Loc, and Other. However, sometimes it is ambiguous whether a word is a person, location, or something else. (Proper nouns are particularly difficult to deal with because they belong to an **open class**, that is, there is an unbounded number of possible names, unlike the set of nouns and verbs, which is large but essentially fixed.) We can get better performance by considering long-range correlations between words. For example, we might add a link between all occurrences of the same word, and force the word to have the same tag in each occurence. (The same technique can also be helpful for resolving the identity of pronouns.) This is known as a **skip-chain CRF**. See Figure 19.17 for an illustration.

We see that the graph structure itself changes depending on the input, which is an additional advantage of CRFs over generative models. Unfortunately, inference in this model is generally more expensive than in a simple chain with local connections, for reasons explained in Section 20.5.

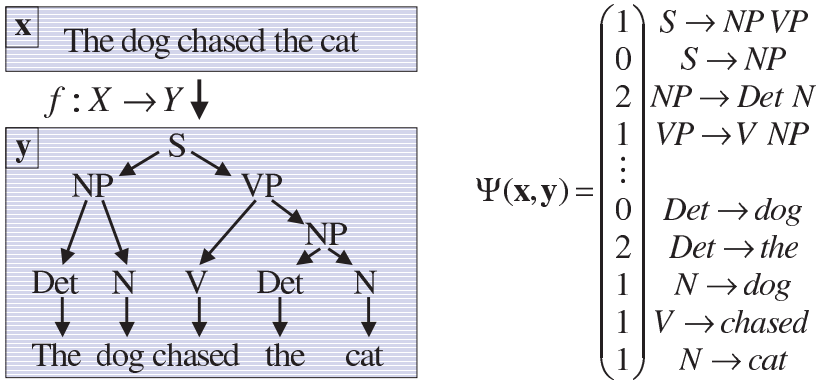


Figure 19.18 Illustration of a simple parse tree based on a context free grammar in Chomsky normal form. The feature vector $\boldsymbol{\phi}(\mathbf{x}, \mathbf{y}) = \Psi(\mathbf{x}, \mathbf{y})$ counts the number of times each production rule was used. Source: Figure 5.2 of (Altun et al. 2006) . Used with kind permission of Yasemin Altun.

19.6.2.4 Natural language parsing

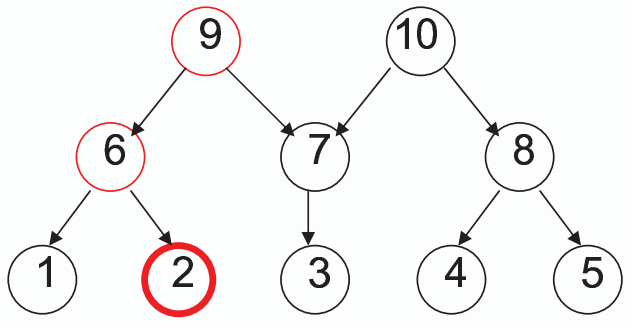
A generalization of chain-structured models for language is to use probabilistic grammars. In particular, a probabilistic **context free grammar** or **PCFG** is a set of re-write or production rules of the form $\sigma \rightarrow \sigma'\sigma''$ or $\sigma \rightarrow x$, where $\sigma, \sigma', \sigma'' \in \Sigma$ are non-terminals (analogous to parts of speech), and $x \in \mathcal{X}$ are terminals, i.e., words. See Figure 19.18 for an example. Each such rule has an associated probability. The resulting model defines a probability distribution over sequences of words. We can compute the probability of observing a particular sequence $\mathbf{x} = x_1 \ldots x_T$ by summing over all trees that generate it. This can be done in $O(T^3)$ time using the **inside-outside algorithm**; see e.g., (Jurafsky and Martin 2008; Manning and Schuetze 1999) for details.

PCFGs are generative models. It is possible to make discriminative versions which encode the probability of a labeled tree, $\mathbf{y}$, given a sequence of words, $\mathbf{x}$, by using a CRF of the form $p(\mathbf{y}|\mathbf{x}) \propto \exp(\mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}, \mathbf{y}))$. For example, we might define $\boldsymbol{\phi}(\mathbf{x}, \mathbf{y})$ to count the number of times each production rule was used (which is analogous to the number of state transitions in a chain-structured model). See e.g., (Taskar et al. 2004) for details.

19.6.2.5 Hierarchical classification

Suppose we are performing multi-class classification, where we have a **label taxonomy**, which groups the classes into a hierarchy. We can encode the position of y within this hierarchy by defining a binary vector $\boldsymbol{\phi}(y)$, where we turn on the bit for component y and for all its children. This can be combined with input features $\boldsymbol{\phi}(\mathbf{x})$ using a tensor product, $\boldsymbol{\phi}(\mathbf{x}, y) = \boldsymbol{\phi}(\mathbf{x}) \otimes \boldsymbol{\phi}(y)$. See Figure 19.19 for an example.

This method is widely used for text classification, where manually constructed taxnomies (such as the Open Directory Project at `www.dmoz.org`) are quite common. The benefit is that information can be shared between the parameters for nearby categories, enabling generalization across classes.



$$\Lambda(2) = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} \quad \Psi(\mathbf{x}, 2) = \begin{pmatrix} 0 \\ \mathbf{x} \\ 0 \\ 0 \\ 0 \\ \mathbf{x} \\ 0 \\ 0 \\ \mathbf{x} \\ 0 \end{pmatrix}$$

$$\langle \mathbf{w}, \Psi(\mathbf{x}, 2) \rangle = \langle w_2, \mathbf{x} \rangle + \langle w_6, \mathbf{x} \rangle + \langle w_9, \mathbf{x} \rangle$$

Figure 19.19 Illustration of a simple label taxonomy, and how it can be used to compute a distributed representation for the label for class 2. In this figure, $\boldsymbol{\phi}(\mathbf{x}) = \mathbf{x}$, $\boldsymbol{\phi}(y = 2) = \Lambda(2)$, $\boldsymbol{\phi}(\mathbf{x}, y)$ is denoted by $\Psi(\mathbf{x}, 2)$, and $\mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}, y)$ is denoted by $\langle \mathbf{w}, \Psi(\mathbf{x}, 2) \rangle$. Source: Figure 5.1 of (Altun et al. 2006) . Used with kind permission of Yasemin Altun.

19.6.2.6 Protein side-chain prediction

An interesting analog to the skip-chain model arises in the problem of predicting the structure of protein side chains. Each residue in the side chain has 4 dihedral angles, which are usually discretized into 3 values called rotamers. The goal is to predict this discrete sequence of angles, $\mathbf{y}$, from the discrete sequence of amino acids, $\mathbf{x}$.

We can define an energy function $E(\mathbf{x}, \mathbf{y})$, where we include various pairwise interaction terms between nearby residues (elements of the $\mathbf{y}$ vector). This energy is usually defined as a weighted sum of individual energy terms, $E(\mathbf{x}, \mathbf{y}|\mathbf{w}) = \sum_{j=1}^{D} \theta_j E_j(\mathbf{x}, \mathbf{y})$, where the E_j are energy contribution due to various electrostatic charges, hydrogen bonding potentials, etc, and $\mathbf{w}$ are the parameters of the model. See (Yanover et al. 2007) for details.

Given the model, we can compute the most probable side chain configuration using $\mathbf{y}^* = \operatorname{argmin} E(\mathbf{x}, \mathbf{y}|\mathbf{w})$. In general, this problem is NP-hard, depending on the nature of the graph induced by the E_j terms, due to long-range connections between the variables. Nevertheless, some special cases can be efficiently handled, using methods discussed in Section 22.6.

19.6.2.7 Stereo vision

Low-level vision problems are problems where the input is an image (or set of images), and the output is a processed version of the image. In such cases, it is common to use 2d lattice-structured models; the models are similar to Figure 19.9, except that the features can be global, and are not generated by the model. We will assume a pairwise CRF.

A classic low-level vision problem is **dense stereo reconstruction**, where the goal is to estimate the depth of every pixel given two images taken from slightly different angles. In this section (based on (Sudderth and Freeman 2008)), we give a sketch of how a simple CRF can be used to solve this task. See e.g., (Sun et al. 2003) for a more sophisticated model.

By using some standard preprocessing techniques, one can convert depth estimation into a

problem of estimating the **disparity** y_s between the pixel at location (i_s, j_s) in the left image and the corresponding pixel at location $(i_s + y_s, j_s)$ in the right image. We typically assume that corresponding pixels have similar intensity, so we define a local node potential of the form

$$\psi_s(y_s|\mathbf{x}) \propto \exp\left\{-\frac{1}{2\sigma^2}\left(x_L(i_s, j_s) - x_R(i_s + y_s, j_s)\right)^2\right\} \tag{19.68}$$

where x_L is the left image and x_R is the right image. This equation can be generalized to model the intensity of small windows around each location. In highly textured regions, it is usually possible to find the corresponding patch using cross correlation, but in regions of low texture, there will be considerable ambiguity about the correct value of y_s.

We can easily add a Gaussian prior on the edges of the MRF that encodes the assumption that neighboring disparities y_s, y_t should be similar, as follows:

$$\psi_{st}(y_s, y_t) \propto \exp\left(-\frac{1}{2\gamma^2}(y_s - y_t)^2\right) \tag{19.69}$$

The resulting model is a Gaussian CRF.

However, using Gaussian edge-potentials will oversmooth the estimate, since this prior fails to account for the occasional large changes in disparity that occur between neighboring pixels which are on different sides of an occlusion boundary. One gets much better results using a **truncated Gaussian potential** of the form

$$\psi_{st}(y_s, y_t) \propto \exp\left\{-\frac{1}{2\gamma^2}\min\left((y_s - y_t)^2, \delta_0^2\right)\right\} \tag{19.70}$$

where γ encodes the expected smoothness, and δ_0 encodes the maximum penalty that will be imposed if disparities are significantly different. This is called a **discontinuity preserving** potential; note that such penalties are not convex. The local evidence potential can be made robust in a similar way, in order to handle outliers due to specularities, occlusions, etc.

Figure 19.20 illustrates the difference between these two forms of prior. On the top left is an image from the standard Middlebury stereo benchmark dataset (Scharstein and Szeliski 2002). On the bottom left is the corresponding true disparity values. The remaining columns represent the estimated disparity after 0, 1 and an "infinite" number of rounds of loopy belief propagation (see Section 22.2), where by "infinite" we mean the results at convergence. The top row shows the results using a Gaussian edge potential, and the bottom row shows the results using the truncated potential. The latter is clearly better.

Unfortunately, performing inference with real-valued variables is computationally difficult, unless the model is jointly Gaussian. Consequently, it is common to discretize the variables. (For example, Figure 19.20(bottom) used 50 states.) The edge potentials still have the form given in Equation 19.69. The resulting model is called a **metric CRF**, since the potentials form a metric. [9] Inference in metric CRFs is more efficient than in CRFs where the discrete labels have no natural ordering, as we explain in Section 22.6.3.3. See Section 22.6.4 for a comparison of various approximate inference methods applied to low-level CRFs, and see (Blake et al. 2011; Prince 2012) for more details on probabilistic models for computer vision.

9. A function f is said to be a **metric** if it satisfies the following three properties: Reflexivity: $f(a, b) = 0$ iff $a = b$; Symmetry: $f(a, b) = f(b, a)$; and Triangle inequality: $f(a, b) + f(b, c) \geq f(a, c)$. If f satisfies only the first two properties, it is called a **semi-metric**.

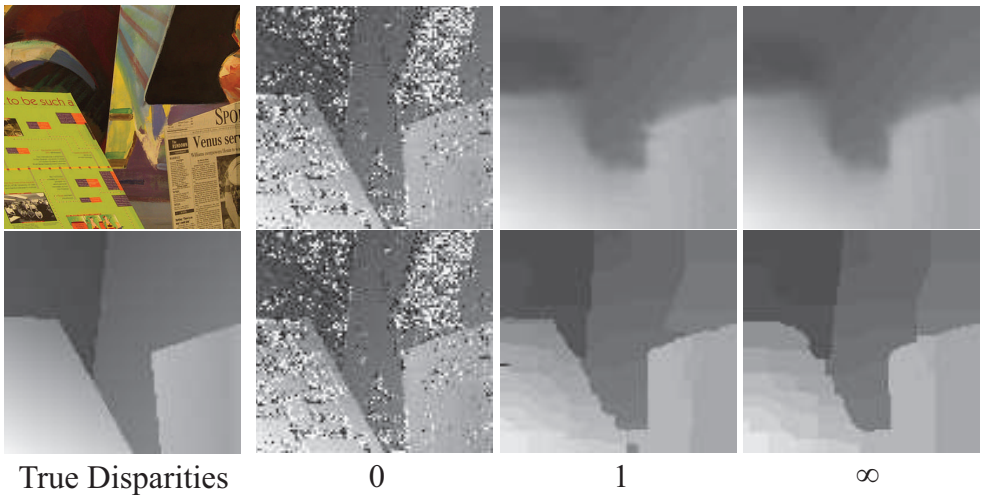


Figure 19.20 Illustration of belief propagation for stereo depth estimation. Left column: image and true disparities. Remaining columns: initial estimate, estimate after 1 iteration, and estimate at convergence. Top row: Gaussian edge potentials. Bottom row: robust edge potentials. Source: Figure 4 of (Sudderth and Freeman 2008). Used with kind permission of Erik Sudderth.

19.6.3 CRF training

We can modify the gradient based optimization of MRFs described in Section 19.5.1 to the CRF case in a straightforward way. In particular, the scaled log-likelihood becomes

$$\ell(\mathbf{w}) \triangleq \frac{1}{N}\sum_i \log p(\mathbf{y}_i|\mathbf{x}_i,\mathbf{w}) = \frac{1}{N}\sum_i \left[\sum_c \mathbf{w}_c^T \boldsymbol{\phi}_c(\mathbf{y}_i,\mathbf{x}_i) - \log Z(\mathbf{w},\mathbf{x}_i)\right] \tag{19.71}$$

and the gradient becomes

$$\frac{\partial \ell}{\partial \mathbf{w}_c} = \frac{1}{N}\sum_i \left[\boldsymbol{\phi}_c(\mathbf{y}_i,\mathbf{x}_i) - \frac{\partial}{\partial \mathbf{w}_c}\log Z(\mathbf{w},\mathbf{x}_i)\right] \tag{19.72}$$

$$= \frac{1}{N}\sum_i \left[\boldsymbol{\phi}_c(\mathbf{y}_i,\mathbf{x}_i) - \mathbb{E}\left[\boldsymbol{\phi}_c(\mathbf{y},\mathbf{x}_i)\right]\right] \tag{19.73}$$

Note that we now have to perform inference for every single training case inside each gradient step, which is $O(N)$ times slower than the MRF case. This is because the partition function depends on the inputs $\mathbf{x}_i$.

In most applications of CRFs (and some applications of MRFs), the size of the graph structure can vary. Hence we need to use parameter tying to ensure we can define a distribution of arbitrary size. In the pairwise case, we can write the model as follows:

$$p(\mathbf{y}|\mathbf{x},\mathbf{w}) = \frac{1}{Z(\mathbf{w},\mathbf{x})}\exp\left(\mathbf{w}^T\boldsymbol{\phi}(\mathbf{y},\mathbf{x})\right) \tag{19.74}$$

where $\mathbf{w} = [\mathbf{w}_n, \mathbf{w}_e]$ are the node and edge parameters, and

$$\boldsymbol{\phi}(\mathbf{y}, \mathbf{x}) \triangleq [\sum_t \boldsymbol{\phi}_t(y_t, \mathbf{x}), \sum_{s \sim t} \boldsymbol{\phi}_{st}(y_s, y_t, \mathbf{x})] \tag{19.75}$$

are the summed node and edge features (these are the sufficient statistics). The gradient expression is easily modified to handle this case.

In practice, it is important to use a prior/ regularization to prevent overfitting. If we use a Gaussian prior, the new objective becomes

$$\ell'(\mathbf{w}) \triangleq \frac{1}{N} \sum_i \log p(\mathbf{y}_i | \mathbf{x}_i, \mathbf{w}) - \lambda ||\mathbf{w}||_2^2 \tag{19.76}$$

It is simple to modify the gradient expression.

Alternatively, we can use ℓ_1 regularization. For example, we could use ℓ_1 for the edge weights $\mathbf{w}_e$ to learn a sparse graph structure, and ℓ_2 for the node weights $\mathbf{w}_n$, as in (Schmidt et al. 2008). In other words, the objective becomes

$$\ell'(\mathbf{w}) \triangleq \frac{1}{N} \sum_i \log p(\mathbf{y}_i | \mathbf{x}_i, \mathbf{w}) - \lambda_1 ||\mathbf{w}_e||_1 - \lambda_2 ||\mathbf{w}_n||_2^2 \tag{19.77}$$

Unfortunately, the optimization algorithms are more complicated when we use ℓ_1 (see Section 13.4), although the problem is still convex.

To handle large datasets, we can use stochastic gradient descent (SGD), as described in Section 8.5.2.

It is possible (and useful) to define CRFs with hidden variables, for example to allow for an unknown alignment between the visible features and the hidden labels (see e.g., (Schnitzspan et al. 2010)). In this case, the objective function is no longer convex. Nevertheless, we can find a locally optimal ML or MAP parameter estimate using EM and/ or gradient methods.

19.7 Structural SVMs

We have seen that training a CRF requires inference, in order to compute the expected sufficient statistics needed to evaluate the gradient. For certain models, computing a joint MAP estimate of the states is provably simpler than computing marginals, as we discuss in Section 22.6. In this section, we discuss a way to train structured output classifiers that that leverages the existence of fast MAP solvers. (To avoid confusion with MAP estimation of parameters, we will often refer to MAP estimation of states as **decoding**.) These methods are known as **structural support vector machines** or **SSVMs** (Tsochantaridis et al. 2005). (There is also a very similar class of methods known as **max margin Markov networks** or **M3nets** (Taskar et al. 2003); see Section 19.7.2 for a discussion of the differences.)

19.7.1 SSVMs: a probabilistic view

In this book, we have mostly concentrated on fitting models using MAP parameter estimation, i.e., by minimizing functions of the form

$$R_{MAP}(\mathbf{w}) = -\log p(\mathbf{w}) - \sum_{i=1}^{N} \log p(\mathbf{y}_i | \mathbf{x}_i, \mathbf{w}) \tag{19.78}$$