19 *Undirected graphical models (Markov random fields)*

19.1 Introduction

In Chapter 10, we discussed directed graphical models (DGMs), commonly known as Bayes nets. However, for some domains, being forced to choose a direction for the edges, as required by a DGM, is rather awkward. For example, consider modeling an image. We might suppose that the intensity values of neighboring pixels are correlated. We can create a DAG model with a 2d lattice topology as shown in Figure 19.1(a). This is known as a **causal MRF** or a **Markov mesh** (Abend et al. 1965). However, its conditional independence properties are rather unnatural. In particular, the Markov blanket (defined in Section 10.5) of the node X_8 in the middle is the other colored nodes (3, 4, 7, 9, 12 and 13) rather than just its 4 nearest neighbors as one might expect.

An alternative is to use an **undirected graphical model** (**UGM**), also called a **Markov random field** (**MRF**) or **Markov network**. These do not require us to specify edge orientations, and are much more natural for some problems such as image analysis and spatial statistics. For example, an undirected 2d lattice is shown in Figure 19.1(b); now the Markov blanket of each node is just its nearest neighbors, as we show in Section 19.2.

Roughly speaking, the main advantages of UGMs over DGMs are: (1) they are symmetric and therefore more "natural" for certain domains, such as spatial or relational data; and (2) discriminativel UGMs (aka conditional random fields, or CRFs), which define conditional densities of the form $p(\mathbf{y}|\mathbf{x})$, work better than discriminative DGMs, for reasons we explain in Section 19.6.1. The main disadvantages of UGMs compared to DGMs are: (1) the parameters are less interpretable and less modular, for reasons we explain in Section 19.3; and (2) parameter estimation is computationally more expensive, for reasons we explain in Section 19.5. See (Domke et al. 2008) for an empirical comparison of the two approaches for an image processing task.

19.2 Conditional independence properties of UGMs

19.2.1 Key properties

UGMs define CI relationships via simple graph separation as follows: for sets of nodes A, B, and C, we say $\mathbf{x}_A \perp_G \mathbf{x}_B|\mathbf{x}_C$ iff C separates A from B in the graph G. This means that, when we remove all the nodes in C, if there are no paths connecting any node in A to any node in B, then the CI property holds. This is called the **global Markov property** for UGMs. For example, in Figure 19.2(b), we have that $\{1, 2\} \perp \{6, 7\}|\{3, 4, 5\}$.

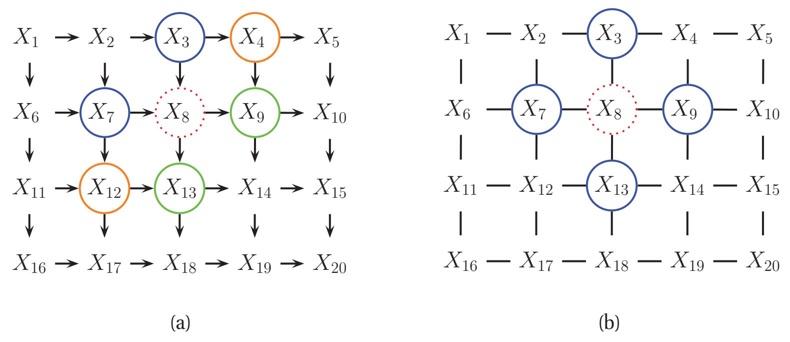


Figure 19.1 (a) A 2d lattice represented as a DAG. The dotted red node X_8 is independent of all other nodes (black) given its Markov blanket, which include its parents (blue), children (green) and co-parents (orange). (b) The same model represented as a UGM. The red node X_8 is independent of the other black nodes given its neighbors (blue nodes).

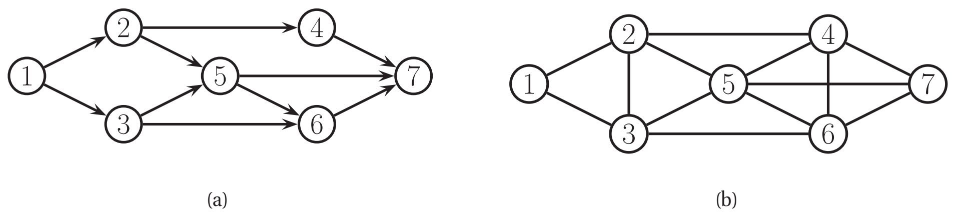


Figure 19.2 (a) A DGM. (b) Its moralized version, represented as a UGM.

The set of nodes that renders a node t conditionally independent of all the other nodes in the graph is called t's **Markov blanket**; we will denote this by $\text{mb}(t)$. Formally, the Markov blanket satisfies the following property:

$$t \perp \mathcal{V} \setminus \text{cl}(t) | \text{mb}(t) \tag{19.1}$$

where $\text{cl}(t) \triangleq \text{mb}(t) \cup \{t\}$ is the **closure** of node t. One can show that, in a UGM, a node's Markov blanket is its set of immediate neighbors. This is called the **undirected local Markov property**. For example, in Figure 19.2(b), we have $\text{mb}(5) = \{2, 3, 4, 6\}$.

From the local Markov property, we can easily see that two nodes are conditionally independent given the rest if there is no direct edge between them. This is called the **pairwise Markov property**. In symbols, this is written as

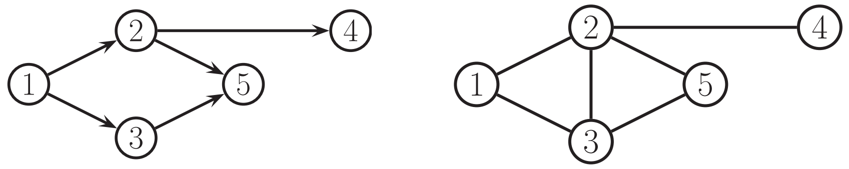
$$s \perp t | \mathcal{V} \setminus \{s, t\} \iff G_{st} = 0 \tag{19.2}$$

Using the three Markov properties we have discussed, we can derive the following CI properties (amongst others) from the UGM in Figure 19.2(b):

- **Pairwise** $1 \perp 7 | \text{rest}$
- **Local** $1 \perp \text{rest} | 2, 3$

$$\begin{tikzcd} G \arrow[r, Rightarrow] & L \arrow[r, Rightarrow] & P \arrow[ll, Rightarrow, bend left, "p(x) > 0"] \end{tikzcd}$$

Figure 19.3 Relationship between Markov properties of UGMs.



(a) (b)

Figure 19.4 (a) The ancestral graph induced by the DAG in Figure 19.2(a) wrt $U = \{2, 4, 5\}$. (b) The moralized version of (a).

- **Global** $1, 2 \perp 6, 7 | 3, 4, 5$

It is obvious that global Markov implies local Markov which implies pairwise Markov. What is less obvious, but nevertheless true (assuming $p(\mathbf{x}) > 0$ for all $\mathbf{x}$, i.e., that p is a positive density), is that pairwise implies global, and hence that all these Markov properties are the same, as illustrated in Figure 19.3 (see e.g., (Koller and Friedman 2009, p119) for a proof).[1] The importance of this result is that it is usually easier to empirically assess pairwise conditional independence; such pairwise CI statements can be used to construct a graph from which global CI statements can be extracted.

19.2.2 An undirected alternative to d-separation

We have seen that determinining CI relationships in UGMs is much easier than in DGMs, because we do not have to worry about the directionality of the edges. In this section, we show how to determine CI relationships for a DGM using a UGM.

It is tempting to simply convert the DGM to a UGM by dropping the orientation of the edges, but this is clearly incorrect, since a v-structure $A \rightarrow B \leftarrow C$ has quite different CI properties than the corresponding undirected chain $A - B - C$. The latter graph incorrectly states that $A \perp C | B$. To avoid such incorrect CI statements, we can add edges between the "unmarried" parents A and C, and then drop the arrows from the edges, forming (in this case) a fully connected undirected graph. This process is called **moralization**. Figure 19.2(b) gives a larger

1. The restriction to positive densities arises because deterministic constraints can result in independencies present in the distribution that are not explicitly represented in the graph. See e.g., (Koller and Friedman 2009, p120) for some examples. Distributions with non-graphical CI properties are said to be **unfaithful** to the graph, so $I(p) \neq I(G)$.

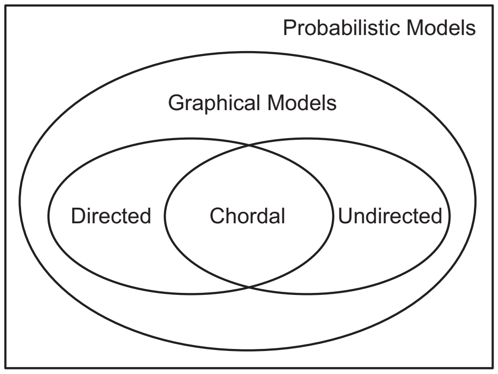


Figure 19.5 DGMs and UGMs can perfectly represent different sets of distributions. Some distributions can be perfectly represented by either DGMs or UGMs; the corresponding graph must be chordal.

example of moralization: we interconnect 2 and 3, since they have a common child 5, and we interconnect 4, 5 and 6, since they have a common child 7.

Unfortunately, moralization loses some CI information, and therefore we cannot use the moralized UGM to determine CI properties of the DGM. For example, in Figure 19.2(a), using d-separation, we see that $4 \perp 5|2$. Adding a moralization arc $4 - 5$ would lose this fact (see Figure 19.2(b)). However, notice that the 4-5 moralization edge, due to the common child 7, is not needed if we do not observe 7 or any of its descendants. This suggests the following approach to determining if $A \perp B|C$. First we form the **ancestral graph** of DAG G with respect to $U = A \cup B \cup C$. This means we remove all nodes from G that are not in U or are not ancestors of U. We then moralize this ancestral graph, and apply the simple graph separation rules for UGMs. For example, in Figure 19.4(a), we show the ancestral graph for Figure 19.2(a) using $U = \{2, 4, 5\}$. In Figure 19.4(b), we show the moralized version of this graph. It is clear that we now correctly conclude that $4 \perp 5|2$.

19.2.3 Comparing directed and undirected graphical models

Which model has more "expressive power", a DGM or a UGM? To formalize this question, recall that we say that G is an I-map of a distribution p if $I(G) \subseteq I(p)$. Now define G to be **perfect map** of p if $I(G) = I(p)$, in other words, the graph can represent all (and only) the CI properties of the distribution. It turns out that DGMs and UGMs are perfect maps for different sets of distributions (see Figure 19.5). In this sense, neither is more powerful than the other as a representation language.

As an example of some CI relationships that can be perfectly modeled by a DGM but not a UGM, consider a v-structure $A \rightarrow C \leftarrow B$. This asserts that $A \perp B$, and $A \not\perp B|C$. If we drop the arrows, we get $A - C - B$, which asserts $A \perp B|C$ and $A \not\perp B$, which is incorrect. In fact, there is no UGM that can precisely represent all and only the two CI statements encoded by a v-structure. In general, CI properties in UGMs are monotonic, in the following sense: if $A \perp B|C$, then $A \perp B|(C \cup D)$. But in DGMs, CI properties can be non-monotonic, since conditioning

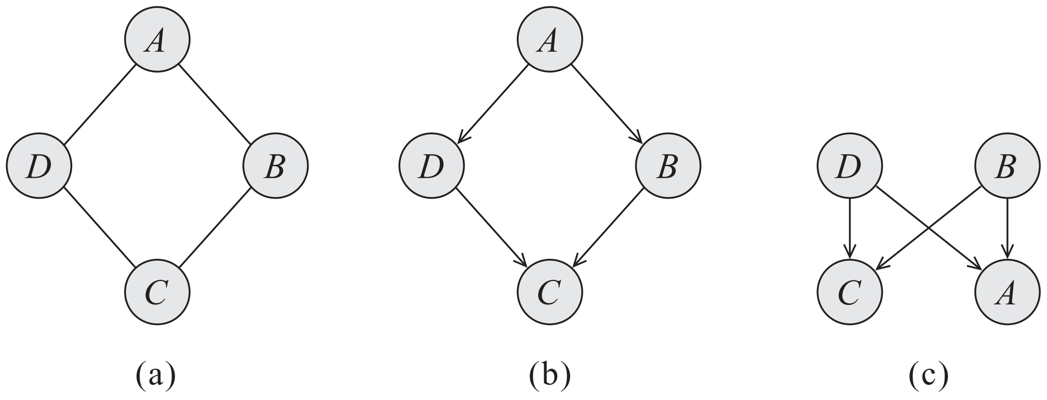


Figure 19.6 A UGM and two failed attempts to represent it as a DGM. Source: Figure 3.10 of (Koller and Friedman 2009). Used with kind permission of Daphne Koller.

on extra variables can eliminate conditional independencies due to explaining away.

As an example of some CI relationships that can be perfectly modeled by a UGM but not a DGM, consider the 4-cycle shown in Figure 19.6(a). One attempt to model this with a DGM is shown in Figure 19.6(b). This correctly asserts that $A \perp C|B, D$. However, it incorrectly asserts that $B \perp D|A$. Figure 19.6(c) is another incorrect DGM: it correctly encodes $A \perp C|B, D$, but incorrectly encodes $B \perp D$. In fact there is no DGM that can precisely represent all and only the CI statements encoded by this UGM.

Some distributions can be perfectly modeled by either a DGM or a UGM; the resulting graphs are called **decomposable** or **chordal**. Roughly speaking, this means the following: if we collapse together all the variables in each maximal clique, to make "mega-variables", the resulting graph will be a tree. Of course, if the graph is already a tree (which includes chains as a special case), it will be chordal. See Section 20.4.1 for further details.

19.3 Parameterization of MRFs

Although the CI properties of UGM are simpler and more natural than for DGMs, representing the joint distribution for a UGM is less natural than for a DGM, as we see below.

19.3.1 The Hammersley-Clifford theorem

Since there is no topological ordering associated with an undirected graph, we can't use the chain rule to represent $p(\mathbf{y})$. So instead of associating CPDs with each node, we associate **potential functions** or **factors** with each maximal clique in the graph. We will denote the potential function for clique c by $\psi_c(\mathbf{y}_c|\boldsymbol{\theta}_c)$. A potential function can be any non-negative function of its arguments. The joint distribution is then defined to be proportional to the product of clique potentials. Rather surprisingly, one can show that any positive distribution whose CI properties can be represented by a UGM can be represented in this way. We state this result more formally below.

Theorem 19.3.1 (**Hammersley-Clifford**). *A positive distribution $p(\mathbf{y}) > 0$ satisfies the CI properties of an undirected graph G iff p can be represented as a product of factors, one per maximal clique, i.e.,*

$$p(\mathbf{y}|\boldsymbol{\theta}) = \frac{1}{Z(\boldsymbol{\theta})} \prod_{c \in \mathcal{C}} \psi_c(\mathbf{y}_c|\boldsymbol{\theta}_c) \tag{19.3}$$

where $\mathcal{C}$ is the set of all the (maximal) cliques of G, and $Z(\boldsymbol{\theta})$ is the **partition function** *given by*

$$Z(\boldsymbol{\theta}) \triangleq \sum_{\mathbf{x}} \prod_{c \in \mathcal{C}} \psi_c(\mathbf{y}_c|\boldsymbol{\theta}_c) \tag{19.4}$$

Note that the partition function is what ensures the overall distribution sums to 1.[2]

The proof was never published, but can be found in e.g., (Koller and Friedman 2009).

For example, consider the MRF in Figure 10.1(b). If p satisfies the CI properties of this graph then we can write p as follows:

$$p(\mathbf{y}|\boldsymbol{\theta}) = \frac{1}{Z(\boldsymbol{\theta})} \psi_{123}(y_1, y_2, y_3)\psi_{234}(y_2, y_3, y_4)\psi_{35}(y_3, y_5) \tag{19.5}$$

where

$$Z = \sum_{\mathbf{y}} \psi_{123}(y_1, y_2, y_3)\psi_{234}(y_2, y_3, y_4)\psi_{35}(y_3, y_5) \tag{19.6}$$

There is a deep connection between UGMs and statistical physics. In particular, there is a model known as the **Gibbs distribution**, which can be written as follows:

$$p(\mathbf{y}|\boldsymbol{\theta}) = \frac{1}{Z(\boldsymbol{\theta})} \exp(-\sum_c E(\mathbf{y}_c|\boldsymbol{\theta}_c)) \tag{19.7}$$

where $E(\mathbf{y}_c) > 0$ is the energy associated with the variables in clique c. We can convert this to a UGM by defining

$$\psi_c(\mathbf{y}_c|\boldsymbol{\theta}_c) = \exp(-E(\mathbf{y}_c|\boldsymbol{\theta}_c)) \tag{19.8}$$

We see that high probability states correspond to low energy configurations. Models of this form are known as **energy based models**, and are commonly used in physics and biochemistry, as well as some branches of machine learning (LeCun et al. 2006).

Note that we are free to restrict the parameterization to the edges of the graph, rather than the maximal cliques. This is called a **pairwise MRF**. In Figure 10.1(b), we get

$$\begin{aligned} p(\mathbf{y}|\boldsymbol{\theta}) &\propto \psi_{12}(y_1, y_2)\psi_{13}(y_1, y_3)\psi_{23}(y_2, y_3)\psi_{24}(y_2, y_4)\psi_{34}(y_3, y_4)\psi_{35}(y_3, y_5) && (19.9) \\ &\propto \prod_{s \sim t} \psi_{st}(y_s, y_t) && (19.10) \end{aligned}$$

This form is widely used due to its simplicity, although it is not as general.

2. The partition function is denoted by Z because of the German word *zustandssumme*, which means “sum over states”. This reflects the fact that a lot of pioneering working in statistical physics was done by Germans.

19.3.2 Representing potential functions

If the variables are discrete, we can represent the potential or energy functions as tables of (non-negative) numbers, just as we did with CPTs. However, the potentials are not probabilities. Rather, they represent the relative "compatibility" between the different assignments to the potential. We will see some examples of this below.

A more general approach is to define the log potentials as a linear function of the parameters:

$$\log \psi_c(\mathbf{y}_c) \triangleq \boldsymbol{\phi}_c(\mathbf{y}_c)^T \boldsymbol{\theta}_c \tag{19.11}$$

where $\boldsymbol{\phi}_c(\mathbf{x}_c)$ is a feature vector derived from the values of the variables $\mathbf{y}_c$. The resulting log probability has the form

$$\log p(\mathbf{y}|\boldsymbol{\theta}) = \sum_c \boldsymbol{\phi}_c(\mathbf{y}_c)^T \boldsymbol{\theta}_c - Z(\boldsymbol{\theta}) \tag{19.12}$$

This is also known as a **maximum entropy** or a **log-linear** model.

For example, consider a pairwise MRF, where for each edge, we associate a feature vector of length K^2 as follows:

$$\boldsymbol{\phi}_{st}(y_s, y_t) = [\ldots, \mathbb{I}(y_s = j, y_t = k), \ldots] \tag{19.13}$$

If we have a weight for each feature, we can convert this into a $K \times K$ potential function as follows:

$$\psi_{st}(y_s = j, y_t = k) = \exp([\boldsymbol{\theta}_{st}^T \boldsymbol{\phi}_{st}]_{jk}) = \exp(\theta_{st}(j,k)) \tag{19.14}$$

So we see that we can easily represent tabular potentials using a log-linear form. But the log-linear form is more general.

To see why this is useful, suppose we are interested in making a probabilistic model of English spelling. Since certain letter combinations occur together quite frequently (e.g., "ing"), we will need higher order factors to capture this. Suppose we limit ourselves to letter trigrams. A tabular potential still has $26^3 = 17,576$ parameters in it. However, most of these triples will never occur.

An alternative approach is to define indicator functions that look for certain "special" triples, such as "ing", "qu-", etc. Then we can define the potential on each trigram as follows:

$$\psi(y_{t-1}, y_t, y_{t+1}) = \exp(\sum_k \theta_k \phi_k(y_{t-1}, y_t, y_{t+1})) \tag{19.15}$$

where k indexes the different features, corresponding to "ing", "qu-", etc., and ϕ_k is the corresponding binary **feature function**. By tying the parameters across locations, we can define the probability of a word of any length using

$$p(\mathbf{y}|\boldsymbol{\theta}) \propto \exp(\sum_t \sum_k \theta_k \phi_k(y_{t-1}, y_t, y_{t+1})) \tag{19.16}$$

This raises the question of where these feature functions come from. In many applications, they are created by hand to reflect domain knowledge (we will see examples later), but it is also possible to learn them from data, as we discuss in Section 19.5.6.

19.4 Examples of MRFs

In this section, we show how several popular probability models can be conveniently expressed as UGMs.

19.4.1 Ising model

The **Ising model** is an example of an MRF that arose from statistical physics.[3] It was originally used for modeling the behavior of magnets. In particular, let $y_s \in \{-1, +1\}$ represent the spin of an atom, which can either be spin down or up. In some magnets, called **ferro-magnets**, neighboring spins tend to line up in the same direction, whereas in other kinds of magnets, called **anti-ferromagnets**, the spins "want" to be different from their neighbors.

We can model this as an MRF as follows. We create a graph in the form of a 2D or 3D lattice, and connect neighboring variables, as in Figure 19.1(b). We then define the following pairwise clique potential:

$$\psi_{st}(y_s, y_t) = \begin{pmatrix} e^{w_{st}} & e^{-w_{st}} \\ e^{-w_{st}} & e^{w_{st}} \end{pmatrix} \tag{19.17}$$

Here w_{st} is the coupling strength between nodes s and t. If two nodes are not connected in the graph, we set $w_{st} = 0$. We assume that the weight matrix $\mathbf{W}$ is symmetric, so $w_{st} = w_{ts}$. Often we assume all edges have the same strength, so $w_{st} = J$ (assuming $w_{st} \neq 0$).

If all the weights are positive, $J > 0$, then neighboring spins are likely to be in the same state; this can be used to model ferromagnets, and is an example of an **associative Markov network**. If the weights are sufficiently strong, the corresponding probability distribution will have two modes, corresponding to the all +1's state and the all -1's state. These are called the **ground states** of the system.

If all of the weights are negative, $J < 0$, then the spins want to be different from their neighbors; this can be used to model an anti-ferromagnet, and results in a **frustrated system**, in which not all the constraints can be satisfied at the same time. The corresponding probability distribution will have multiple modes. Interestingly, computing the partition function $Z(J)$ can be done in polynomial time for associative Markov networks, but is NP-hard in general (Cipra 2000).

There is an interesting analogy between Ising models and Gaussian graphical models. First, assuming $y_t \in \{-1, +1\}$, we can write the unnormalized log probability of an Ising model as follows:

$$\log \tilde{p}(\mathbf{y}) = -\sum_{s \sim t} y_s w_{st} y_t = -\frac{1}{2}\mathbf{y}^T \mathbf{W} \mathbf{y} \tag{19.18}$$

(The factor of $\frac{1}{2}$ arises because we sum each edge twice.) If $w_{st} = J > 0$, we get a low energy (and hence high probability) if neighboring states agree.

Sometimes there is an **external field**, which is an energy term which is added to each spin. This can be modelled using a local energy term of the form $-\mathbf{b}^T\mathbf{y}$, where $\mathbf{b}$ is sometimes called

3. Ernst Ising was a German-American physicist, 1900–1998.

a **bias term**. The modified distribution is given by

$$\log \tilde{p}(\mathbf{y}) = \sum_{s \sim t} w_{st} y_s y_t + \sum_s b_s y_s = \frac{1}{2}\mathbf{y}^T \mathbf{W}\mathbf{y} + \mathbf{b}^T \mathbf{y} \tag{19.19}$$

where $\boldsymbol{\theta} = (\mathbf{W}, \mathbf{b})$.

If we define $\boldsymbol{\mu} \triangleq -\frac{1}{2}\boldsymbol{\Sigma}^{-1}\mathbf{b}$, $\boldsymbol{\Sigma}^{-1} = -\mathbf{W}$, and $c \triangleq \frac{1}{2}\boldsymbol{\mu}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}$, we can rewrite this in a form that looks similar to a Gaussian:

$$\tilde{p}(\mathbf{y}) \propto \exp(-\frac{1}{2}(\mathbf{y} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{y} - \boldsymbol{\mu}) + c) \tag{19.20}$$

One very important difference is that, in the case of Gaussians, the normalization constant, $Z = |2\pi\boldsymbol{\Sigma}|$, requires the computation of a matrix determinant, which can be computed in $O(D^3)$ time, whereas in the case of the Ising model, the normalization constant requires summing over all 2^D bit vectors; this is equivalent to computing the matrix permanent, which is NP-hard in general (Jerrum et al. 2004).

19.4.2 Hopfield networks

A **Hopfield network** (Hopfield 1982) is a fully connected Ising model with a symmetric weight matrix, $\mathbf{W} = \mathbf{W}^T$. These weights, plus the bias terms $\mathbf{b}$, can be learned from training data using (approximate) maximum likelihood, as described in Section 19.5.[4]

The main application of Hopfield networks is as an **associative memory** or **content addressable memory**. The idea is this: suppose we train on a set of fully observed bit vectors, corresponding to patterns we want to memorize. Then, at test time, we present a partial pattern to the network. We would like to estimate the missing variables; this is called **pattern completion**. See Figure 19.7 for an example. This can be thought of as retrieving an example from memory based on a piece of the example itself, hence the term "associative memory".

Since exact inference is intractable in this model, it is standard to use a coordinate descent algorithm known as **iterative conditional modes** (ICM), which just sets each node to its most likely (lowest energy) state, given all its neighbors. The full conditional can be shown to be

$$p(y_s = 1|\mathbf{y}_{-s}, \boldsymbol{\theta}) = \text{sigm}(\mathbf{w}_{s,:}^T \mathbf{y}_{-s} + b_s) \tag{19.21}$$

Picking the most probable state amounts to using the rule $y_s^* = 1$ if $\sum_t w_{st} y_t > b_s$ and using $y_s^* = 0$ otherwise. (Much better inference algorithms will be discussed later in this book.)

Since inference is deterministic, it is also possible to interpret this model as a **recurrent neural network**. (This is quite different from the feedforward neural nets studied in Section 16.5; they are univariate conditional density models of the form $p(y|\mathbf{x}, \boldsymbol{\theta})$ which can only be used for supervised learning.) See Hertz et al. (1991) for further details on Hopfield networks.

A **Boltzmann machine** generalizes the Hopfield / Ising model by including some hidden nodes, which makes the model representationally more powerful. Inference in such models often uses Gibbs sampling, which is a stochastic version of ICM (see Section 24.2 for details).

4. ML estimation works much better than the outer product rule proposed in in (Hopfield 1982), because it not only lowers the energy of the observed patterns, but it also raises the energy of the non-observed patterns, in order to make the distribution sum to one (Hillar et al. 2012).

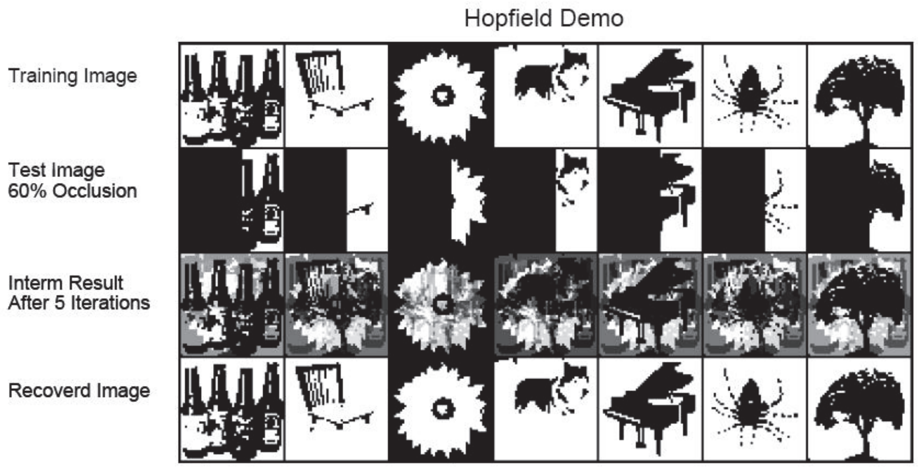


Figure 19.7 Examples of how an associative memory can reconstruct images. These are binary images of size 50×50 pixels. Top: training images. Row 2: partially visible test images. Row 3: estimate after 5 iterations. Bottom: final state estimate. Based on Figure 2.1 of Hertz et al. (1991). Figure generated by `hopfieldDemo`.

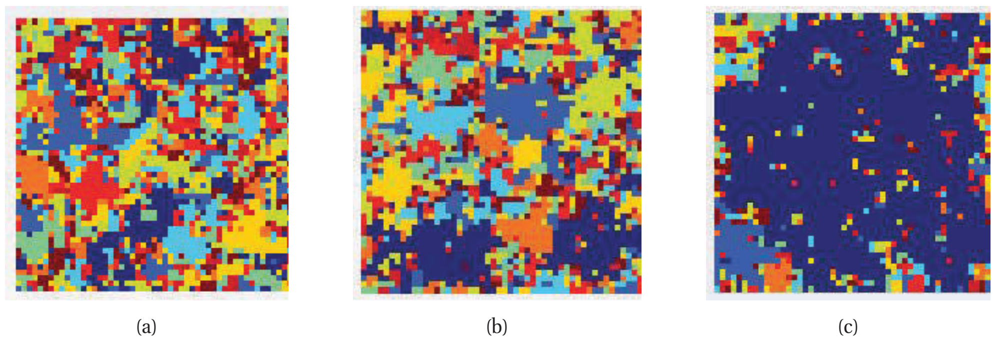


Figure 19.8 Visualizing a sample from a 10-state Potts model of size 128×128 for different association strengths: (a) $J = 1.42$, (b) $J = 1.44$, (c) $J = 1.46$. The regions are labeled according to size: blue is largest, red is smallest. Used with kind permission of Erik Sudderth. See `gibbsDemoIsing` for Matlab code to produce a similar plot for the Ising model.

However, we could equally well apply Gibbs to a Hopfield net and ICM to a Boltzmann machine: the inference algorithm is not part of the model definition. See Section 27.7 for further details on Boltzmann machines.

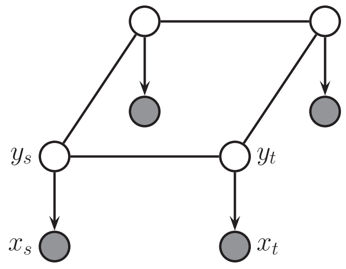


Figure 19.9 A grid-structured MRF with local evidence nodes.

19.4.3 Potts model

It is easy to generalize the Ising model to multiple discrete states, $y_t \in \{1, 2, \ldots, K\}$. It is common to use a potential function of the following form:

$$\psi_{st}(y_s, y_t) = \begin{pmatrix} e^J & 0 & 0 \\ 0 & e^J & 0 \\ 0 & 0 & e^J \end{pmatrix} \tag{19.22}$$

This is called the **Potts model**.[5] If $J > 0$, then neighboring nodes are encouraged to have the same label. Some samples from this model are shown in Figure 19.8. We see that for $J > 1.44$, large clusters occur, for $J < 1.44$, many small clusters occur, and at the **critical value** of $K = 1.44$, there is a mix of small and large clusters. This rapid change in behavior as we vary a parameter of the system is called a **phase transition**, and has been widely studied in the physics community. An analogous phenomenon occurs in the Ising model; see (MacKay 2003, ch 31) for details.

The Potts model can be used as a prior for **image segmentation**, since it says that neighboring pixels are likely to have the same discrete label and hence belong to the same segment. We can combine this prior with a likelihood term as follows:

$$p(\mathbf{y}, \mathbf{x}|\boldsymbol{\theta}) = p(\mathbf{y}|J) \prod_t p(x_t|y_t, \boldsymbol{\theta}) = \left[\frac{1}{Z(J)} \prod_{s \sim t} \psi(y_s, y_t; J) \right] \prod_t p(x_t|y_t, \boldsymbol{\theta}) \tag{19.23}$$

where $p(x_t|y_t = k, \boldsymbol{\theta})$ is the probability of observing pixel x_t given that the corresponding segment belongs to class k. This observation model can be modeled using a Gaussian or a non-parametric density. (Note that we label the hidden nodes y_t and the observed nodes x_t, to be compatible with Section 19.6.)

The corresponding graphical model is a mix of undirected and directed edges, as shown in Figure 19.9. The undirected 2d lattice represents the prior $p(\mathbf{y})$; in addition, there are directed edge from each y_t to its corresponding x_t, representing the **local evidence**. Technically speaking, this combination of an undirected and directed graph is called a **chain graph**. However,

5. Renfrey Potts was an Australian mathematician, 1925–2005.

since the x_t nodes are observed, they can be "absorbed" into the model, thus leaving behind an undirected "backbone".

This model is a 2d analog of an HMM, and could be called a **partially observed MRF**. As in an HMM, the goal is to perform posterior inference, i.e., to compute (some function of) $p(\mathbf{y}|\mathbf{x}, \boldsymbol{\theta})$. Unfortunately, the 2d case is provably much harder than the 1d case, and we must resort to approximate methods, as we discuss in later chapters.

Although the Potts prior is adequate for regularizing supervised learning problems, it is not sufficiently accurate to perform image segmentation in an unsupervised way, since the segments produced by this model do not accurately represent the kinds of segments one sees in natural images (Morris et al. 1996).[6] For the unsupervised case, one needs to use more sophisticated priors, such as the truncated Gaussian process prior of (Sudderth and Jordan 2008).

19.4.4 Gaussian MRFs

An undirected GGM, also called a **Gaussian MRF** (see e.g., (Rue and Held 2005)), is a pairwise MRF of the following form:

$$p(\mathbf{y}|\boldsymbol{\theta}) \propto \prod_{s\sim t} \psi_{st}(y_s, y_t) \prod_t \psi_t(y_t) \tag{19.24}$$

$$\psi_{st}(y_s, y_t) = \exp(-\frac{1}{2} y_s \Lambda_{st} y_t) \tag{19.25}$$

$$\psi_t(y_t) = \exp(-\frac{1}{2}\Lambda_{tt} y_t^2 + \eta_t y_t) \tag{19.26}$$

(Note that we could easily absorb the node potentials ψ_t into the edge potentials, but we have kept them separate for clarity.) The joint distribution can be written as follows:

$$p(\mathbf{y}|\boldsymbol{\theta}) \propto \exp[\boldsymbol{\eta}^T \mathbf{y} - \frac{1}{2}\mathbf{y}^T \boldsymbol{\Lambda}\mathbf{y}] \tag{19.27}$$

We recognize this as a multivariate Gaussian written in **information form** where $\boldsymbol{\Lambda} = \boldsymbol{\Sigma}^{-1}$ and $\boldsymbol{\eta} = \boldsymbol{\Lambda}\boldsymbol{\mu}$.

If $\Lambda_{st} = 0$, then there is no pairwise term connecting s and t, so by the factorization theorem (Theorem 2.2.1), we conclude that

$$y_s \perp y_t | \mathbf{y}_{-(st)} \iff \Lambda_{st} = 0 \tag{19.28}$$

The zero entries in $\boldsymbol{\Lambda}$ are called **structural zeros**, since they represent the absent edges in the graph. Thus undirected GGMs correspond to sparse precision matrices, a fact which we exploit in Section 26.7.2 to efficiently learn the structure of the graph.

19.4.4.1 Comparing Gaussian DGMs and UGMs *

In Section 10.2.5, we saw that directed GGMs correspond to sparse regression matrices, and hence sparse Cholesky factorizations of covariance matrices, whereas undirected GGMs correspond to

6. An influential paper (Geman and Geman 1984), which introduced the idea of a Gibbs sampler (Section 24.2), proposed using the Potts model as a prior for image segmentation, but the results in their paper are misleading because they did not run their Gibbs sampler for long enough. See Figure 24.10 for a vivid illustration of this point.

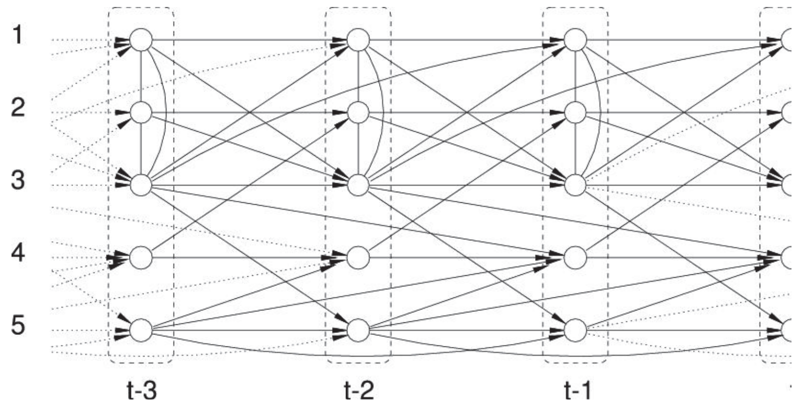


Figure 19.10 A VAR(2) process represented as a dynamic chain graph. Source: (Dahlhaus and Eichler 2000). Used with kind permission of Rainer Dahlhaus and Oxford University Press.

sparse precision matrices. The advantage of the DAG formulation is that we can make the regression weights $\mathbf{W}$, and hence $\mathbf{\Sigma}$, be conditional on covariate information (Pourahmadi 2004), without worrying about positive definite constraints. The disadavantage of the DAG formulation is its dependence on the order, although in certain domains, such as time series, there is already a natural ordering of the variables.

It is actually possible to combine both representations, resulting in a Gaussian chain graph. For example, consider a a discrete-time, second-order Markov chain in which the states are continuous, $\mathbf{y}_t \in \mathbb{R}^D$. The transition function can be represented as a (vector-valued) linear-Gaussian CPD:

$$p(\mathbf{y}_t|\mathbf{y}_{t-1}, \mathbf{y}_{t-2}, \boldsymbol{\theta}) = \mathcal{N}(\mathbf{y}_t|\mathbf{A}_1\mathbf{y}_{t-1} + \mathbf{A}_2\mathbf{y}_{t-2}, \mathbf{\Sigma}) \tag{19.29}$$

This is called **vector auto-regressive** or **VAR** process of order 2. Such models are widely used in econometrics for time-series forecasting.

The time series aspect is most naturally modeled using a DGM. However, if $\mathbf{\Sigma}^{-1}$ is sparse, then the correlation amongst the components within a time slice is most naturally modeled using a UGM. For example, suppose we have

$$\mathbf{A}_1 = \begin{pmatrix} \frac{3}{5} & 0 & \frac{1}{5} & 0 & 0 \\ 0 & \frac{3}{5} & 0 & -\frac{1}{5} & 0 \\ \frac{2}{5} & \frac{1}{3} & \frac{3}{5} & 0 & 0 \\ 0 & 0 & 0 & -\frac{1}{2} & \frac{1}{5} \\ 0 & 0 & \frac{1}{5} & 0 & \frac{2}{5} \end{pmatrix}, \; \mathbf{A}_2 = \begin{pmatrix} 0 & 0 & -\frac{1}{5} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{5} & 0 & \frac{1}{3} \\ 0 & 0 & 0 & 0 & -\frac{1}{5} \end{pmatrix} \tag{19.30}$$

and

$$\mathbf{\Sigma} = \begin{pmatrix} 1 & \frac{1}{2} & \frac{1}{3} & 0 & 0 \\ \frac{1}{2} & 1 & -\frac{1}{3} & 0 & 0 \\ \frac{1}{3} & -\frac{1}{3} & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}, \; \mathbf{\Sigma}^{-1} = \begin{pmatrix} 2.13 & -1.47 & -1.2 & 0 & 0 \\ -1.47 & 2.13 & 1.2 & 0 & 0 \\ -1.2 & 1.2 & 1.8 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \tag{19.31}$$

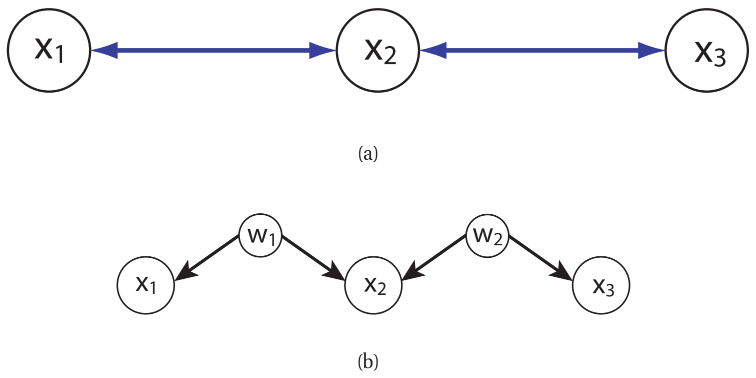


Figure 19.11 (a) A bi-directed graph. (b) The equivalent DAG. Here the w nodes are latent confounders. Based on Figures 5.12-5.13 of (Choi 2011). Used with kind permission of Myung Choi.

The resulting graphical model is illustrated in Figure 19.10. Zeros in the transition matrices $\mathbf{A}_1$ and $\mathbf{A}_2$ correspond to absent directed arcs from $\mathbf{y}_{t-1}$ and $\mathbf{y}_{t-2}$ into $\mathbf{y}_t$. Zeros in the precision matrix $\mathbf{\Sigma}^{-1}$ correspond to absent undirected arcs between nodes in $\mathbf{y}_t$.

Sometimes we have a sparse covariance matrix rather than a sparse precision matrix. This can be represented using a **bi-directed graph**, where each edge has arrows in both directions, as in Figure 19.11(a). Here nodes that are not connected are unconditionally independent. For example in Figure 19.11(a) we see that $Y_1 \perp Y_3$. In the Gaussian case, this means $\Sigma_{1,3} = \Sigma_{3,1} = 0$. (A graph representing a sparse covariance matrix is called a **covariance graph**.) By contrast, if this were an undirected model, we would have that $Y_1 \perp Y_3 | Y_2$, and $\Lambda_{1,3} = \Lambda_{3,1} = 0$, where $\mathbf{\Lambda} = \mathbf{\Sigma}^{-1}$.

A bidirected graph can be converted to a DAG with latent variables, where each bidirected edge is replaced with a hidden variable representing a hidden common cause, or **confounder**, as illustrated in Figure 19.11(b). The relevant CI properties can then be determined using d-separation.

We can combine bidirected and directed edges to get a **directed mixed graphical model**. This is useful for representing a variety of models, such as ARMA models (Section 18.2.4.4), structural equation models (Section 26.5.5), etc.

19.4.5 Markov logic networks *

In Section 10.2.2, we saw how we could “unroll” Markov models and HMMs for an arbitrary number of time steps in order to model variable-length sequences. Similarly, in Section 19.4.1, we saw how we could expand a lattice UGM to model images of any size. What about more complex domains, where we have a variable number of objects and relationships between them? Creating models for such scenarios is often done using **first-order logic** (see e.g., (Russell and Norvig 2010)). For example, consider the sentences “Smoking causes cancer” and “If two people are friends, and one smokes, then so does the other”. We can write these sentences in first-order

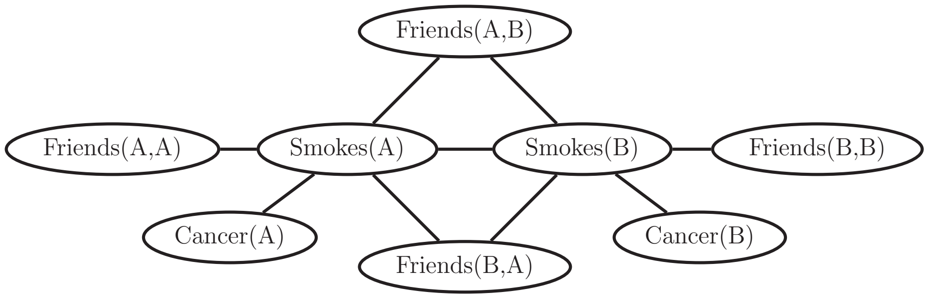


Figure 19.12 An example of a ground Markov logic network represented as a pairwise MRF for 2 people. Based on Figure 2.1 from (Domingos and Lowd 2009). Used with kind permission of Pedro Domingos.

logic as follows:

$$\forall x. Sm(x) \implies Ca(x) \quad (19.32)$$

$$\forall x. \forall y. Fr(x,y) \wedge Sm(x) \implies Sm(y) \quad (19.33)$$

where Sm and Ca are predicates, and Fr is a relation.[7]

Of course, such rules are not always true. Indeed, this brittleness is the main reason why logical approaches to AI are no longer widely used, at least not in their pure form. There have been a variety of attempts to combine first order logic with probability theory, an area known as **statistical relational AI** or **probabilistic relational modeling** (Kersting et al. 2011). One simple approach is to take logical rules and attach weights (known as **certainty factors**) to them, and then to interpret them as conditional probability distributions. For example, we might say $p(Ca(x) = 1|Sm(x) = 1) = 0.9$. Unfortunately, the rule does not say what to predict if $Sm(x) = 0$. Furthermore, combining CPDs in this way is not guaranteed to define a consistent joint distribution, because the resulting graph may not be a DAG.

An alternative approach is to treat these rules as a way of defining potential functions in an unrolled UGM. The result is known as a **Markov logic network** (Domingos and Lowd 2009). To specify the network, we first rewrite all the rules in **conjunctive normal form** (CNF), also known as **clausal form**. In this case, we get

$$\neg Sm(x) \vee Ca(x) \quad (19.34)$$

$$\neg Fr(x,y) \vee \neg Sm(x) \vee Sm(y) \quad (19.35)$$

The first clause can be read as "Either x does not smoke or he has cancer", which is logically equivalent to Equation 19.32. (Note that in a clause, any unbound variable, such as x, is assumed to be universally quantified.)

7. A predicate is just a function of one argument, known as an object, that evaluates to true or false, depending on whether the property holds or not of that object. A (logical) relation is just a function of two or more arguments (objects) that evaluates to true or false, depending on whether the relationship holds between that set of objects or not.

Inference in first-order logic is only semi-decidable, so it is common to use a restricted subset. A common approach (as used in Prolog) is to restrict the language to **Horn clauses**, which are clauses that contain at most one positive literal. Essentially this means the model is a series of if-then rules, where the right hand side of the rules (the "then" part, or consequence) has only a single term.

Once we have encoded our **knowledge base** as a set of clauses, we can attach weights to each one; these weights are the parameter of the model, and they define the clique potentials as follows:

$$\psi_c(\mathbf{x}_c) = \exp(w_c \phi_c(\mathbf{x}_c)) \tag{19.36}$$

where $\phi_c(\mathbf{x}_c)$ is a logical expression which evaluates clause c applied to the variables $\mathbf{x}_c$, and w_c is the weight we attach to this clause. Roughly speaking, the weight of a clause specifies the probability of a world in which this clause is satsified relative to a world in which it is not satisfied.

Now suppose there are two objects (people) in the world, Anna and Bob, which we will denote by **constant symbols** A and B. We can make a **ground network** from the above clauses by creating binary random variables S_x, C_x, and $F_{x,y}$ for $x, y \in \{A, B\}$, and then "wiring these up" according to the clauses above. The result is the UGM in Figure 19.12 with 8 binary nodes. Note that we have not encoded the fact that Fr is a symmetric relation, so $Fr(A, B)$ and $Fr(B, A)$ might have different values. Similarly, we have the "degenerate" nodes $Fr(A, A)$ and $Fr(B, B)$, since we did not enforce $x \neq y$ in Equation 19.33. (If we add such constraints, then the model compiler, which generates the ground network, could avoid creating redundant nodes.)

In summary, we can think of MLNs as a convenient way of specifying a UGM **template**, that can get unrolled to handle data of arbitrary size. There are several other ways to define relational probabilistic models; see e.g., (Koller and Friedman 2009; Kersting et al. 2011) for details. In some cases, there is uncertainty about the number or existence of objects or relations (the so-called **open universe** problem). Section 18.6.2 gives a concrete example in the context of multi-object tracking. See e.g., (Russell and Norvig 2010; Kersting et al. 2011) and references therein for further details.

19.5 Learning

In this section, we discuss how to perform ML and MAP parameter estimation for MRFs. We will see that this is quite computationally expensive. For this reason, it is rare to perform Bayesian inference for the parameters of MRFs (although see (Qi et al. 2005)).

19.5.1 Training maxent models using gradient methods

Consider an MRF in log-linear form:

$$p(\mathbf{y}|\boldsymbol{\theta}) = \frac{1}{Z(\boldsymbol{\theta})} \exp\left(\sum_c \boldsymbol{\theta}_c^T \boldsymbol{\phi}_c(\mathbf{y})\right) \tag{19.37}$$

where c indexes the cliques. The scaled log-likelihood is given by

$$\ell(\boldsymbol{\theta}) \triangleq \frac{1}{N}\sum_i \log p(\mathbf{y}_i|\boldsymbol{\theta}) = \frac{1}{N}\sum_i \left[\sum_c \boldsymbol{\theta}_c^T \boldsymbol{\phi}_c(\mathbf{y}_i) - \log Z(\boldsymbol{\theta})\right] \tag{19.38}$$

Since MRFs are in the exponential family, we know that this function is convex in $\boldsymbol{\theta}$ (see Section 9.2.3), so it has a unique global maximum which we can find using gradient-based optimizers. In particular, the derivative for the weights of a particular clique, c, is given by

$$\frac{\partial \ell}{\partial \boldsymbol{\theta}_c} = \frac{1}{N}\sum_i \left[\boldsymbol{\phi}_c(\mathbf{y}_i) - \frac{\partial}{\partial \boldsymbol{\theta}_c}\log Z(\boldsymbol{\theta})\right] \tag{19.39}$$

Exercise 19.1 asks you to show that the derivative of the log partition function wrt $\boldsymbol{\theta}_c$ is the expectation of the c'th feature under the model, i.e.,

$$\frac{\partial \log Z(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}_c} = \mathbb{E}\left[\boldsymbol{\phi}_c(\mathbf{y})|\boldsymbol{\theta}\right] = \sum_{\mathbf{y}} \boldsymbol{\phi}_c(\mathbf{y}) p(\mathbf{y}|\boldsymbol{\theta}) \tag{19.40}$$

Hence the gradient of the log likelihood is

$$\frac{\partial \ell}{\partial \boldsymbol{\theta}_c} = \left[\frac{1}{N}\sum_i \boldsymbol{\phi}_c(\mathbf{y}_i)\right] - \mathbb{E}\left[\boldsymbol{\phi}_c(\mathbf{y})\right] \tag{19.41}$$

In the first term, we fix $\mathbf{y}$ to its observed values; this is sometimes called the **clamped term**. In the second term, $\mathbf{y}$ is free; this is sometimes called the **unclamped term** or **contrastive term**. Note that computing the unclamped term requires inference in the model, and this must be done once per gradient step. This makes UGM training much slower than DGM training.

The gradient of the log likelihood can be rewritten as the expected feature vector according to the empirical distribution minus the model's expectation of the feature vector:

$$\frac{\partial \ell}{\partial \boldsymbol{\theta}_c} = \mathbb{E}_{p_{\text{emp}}}\left[\boldsymbol{\phi}_c(\mathbf{y})\right] - \mathbb{E}_{p(\cdot|\boldsymbol{\theta})}\left[\boldsymbol{\phi}_c(\mathbf{y})\right] \tag{19.42}$$

At the optimum, the gradient will be zero, so the empirical distribution of the features will match the model's predictions:

$$\mathbb{E}_{p_{\text{emp}}}\left[\boldsymbol{\phi}_c(\mathbf{y})\right] = \mathbb{E}_{p(\cdot|\boldsymbol{\theta})}\left[\boldsymbol{\phi}_c(\mathbf{y})\right] \tag{19.43}$$

This is called **moment matching**. This observation motivates a different optimization algorithm which we discuss in Section 19.5.7.

19.5.2 Training partially observed maxent models

Suppose we have missing data and/or hidden variables in our model. In general, we can represent such models as follows:

$$p(\mathbf{y}, \mathbf{h}|\boldsymbol{\theta}) = \frac{1}{Z(\boldsymbol{\theta})}\exp\left(\sum_c \boldsymbol{\theta}_c^T \boldsymbol{\phi}_c(\mathbf{h}, \mathbf{y})\right) \tag{19.44}$$

The log likelihood has the form

$$\ell(\boldsymbol{\theta}) = \frac{1}{N}\sum_i \log\left(\sum_{\mathbf{h}_i} p(\mathbf{y}_i, \mathbf{h}_i|\boldsymbol{\theta})\right) = \frac{1}{N}\sum_i \log\left(\frac{1}{Z(\boldsymbol{\theta})}\sum_{\mathbf{h}_i} \tilde{p}(\mathbf{y}_i, \mathbf{h}_i|\boldsymbol{\theta})\right) \tag{19.45}$$

where

$$\tilde{p}(\mathbf{y}, \mathbf{h}|\boldsymbol{\theta}) \triangleq \exp\left(\sum_c \boldsymbol{\theta}_c^T \boldsymbol{\phi}_c(\mathbf{h}, \mathbf{y})\right) \tag{19.46}$$

is the unnormalized distribution. The term $\sum_{\mathbf{h}_i} \tilde{p}(\mathbf{y}_i, \mathbf{h}_i|\boldsymbol{\theta})$ is the same as the partition function for the whole model, except that $\mathbf{y}$ is fixed at $\mathbf{y}_i$. Hence the gradient is just the expected features where we clamp $\mathbf{y}_i$, but average over $\mathbf{h}$:

$$\frac{\partial}{\partial \boldsymbol{\theta}_c}\log\left(\sum_{\mathbf{h}_i} \tilde{p}(\mathbf{y}_i, \mathbf{h}_i|\boldsymbol{\theta})\right) = \mathbb{E}\left[\boldsymbol{\phi}_c(\mathbf{h}, \mathbf{y}_i)|\boldsymbol{\theta}\right] \tag{19.47}$$

So the overall gradient is given by

$$\frac{\partial \ell}{\partial \boldsymbol{\theta}_c} = \frac{1}{N}\sum_i \left\{\mathbb{E}\left[\boldsymbol{\phi}_c(\mathbf{h}, \mathbf{y}_i)|\boldsymbol{\theta}\right] - \mathbb{E}\left[\boldsymbol{\phi}_c(\mathbf{h}, \mathbf{y})|\boldsymbol{\theta}\right]\right\} \tag{19.48}$$

The first set of expectations are computed by "clamping" the visible nodes to their observed values, and the second set are computed by letting the visible nodes be free. In both cases, we marginalize over $\mathbf{h}_i$.

An alternative approach is to use generalized EM, where we use gradient methods in the M step. See (Koller and Friedman 2009, p956) for details.

19.5.3 Approximate methods for computing the MLEs of MRFs

When fitting a UGM there is (in general) no closed form solution for the ML or the MAP estimate of the parameters, so we need to use gradient-based optimizers. This gradient requires inference. In models where inference is intractable, learning also becomes intractable. This has motivated various computationally faster alternatives to ML/MAP estimation, which we list in Table 19.1. We dicsuss some of these alternatives below, and defer others to later sections.

19.5.4 Pseudo likelihood

One alternative to MLE is to maximize the **pseudo likelihood** (Besag 1975), defined as follows:

$$\ell_{PL}(\boldsymbol{\theta}) \triangleq \sum_{\mathbf{y}}\sum_{d=1}^{D} p_{\text{emp}}(\mathbf{y} \log p(y_d|\mathbf{y}_{-d}) = \frac{1}{N}\sum_{i=1}^{N}\sum_{d=1}^{D} \log p(y_{id}|\mathbf{y}_{i,-d}, \boldsymbol{\theta}) \tag{19.49}$$

That is, we optimize the product of the full conditionals, also known as the **composite likelihood** (Lindsay 1988), Compare this to the objective for maximum likelihood:

$$\ell_{ML}(\boldsymbol{\theta}) = \sum_{\mathbf{y},\mathbf{x}} p_{\text{emp}}(\mathbf{y} \log p(\mathbf{y}|\boldsymbol{\theta}) = \sum_{i=1}^{N} \log p(\mathbf{y}_i|\boldsymbol{\theta}) \tag{19.50}$$

Method	Restriction	Exact MLE?	Section
Closed form	Only Chordal MRF	Exact	Section 19.5.7.4
IPF	Only Tabular / Gaussian MRF	Exact	Section 19.5.7
Gradient-based optimization	Low tree width	Exact	Section 19.5.1
Max-margin training	Only CRFs	N/A	Section 19.7
Pseudo-likelihood	No hidden variables	Approximate	Section 19.5.4
Stochastic ML	-	Exact (up to MC error)	Section 19.5.5
Contrastive divergence	-	Approximate	Section 27.7.2.4
Minimum probability flow	Can integrate out the hiddens	Approximate	Sohl-Dickstein et al. (2011)

Table 19.1 Some methods that can be used to compute approximate ML/ MAP parameter estimates for MRFs/ CRFs. Low tree-width means that, in order for the method to be efficient, the graph must "tree-like"; see Section 20.5 for details.

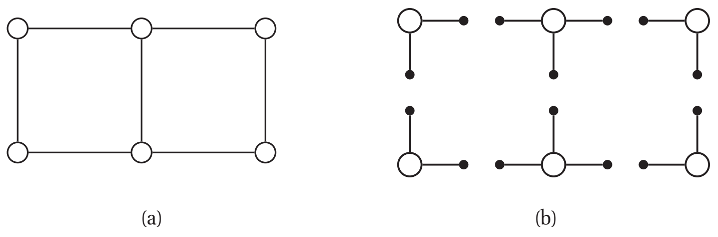


Figure 19.13 (a) A small 2d lattice. (b) The representation used by pseudo likelihood. Solid nodes are observed neighbors. Based on Figure 2.2 of (Carbonetto 2003).

In the case of Gaussian MRFs, PL is equivalent to ML (Besag 1975), but this is not true in general (Liang and Jordan 2008).

The PL approach is illustrated in Figure 19.13 for a 2d grid. We learn to predict each node, given all of its neighbors. This objective is generally fast to compute since each full conditional $p(y_{id}|\mathbf{y}_{i,-d}, \boldsymbol{\theta})$ only requires summing over the states of a single node, y_{id}, in order to compute the local normalization constant. The PL approach is similar to fitting each full conditional separately (which is the method used to train dependency networks, discussed in Section 26.2.2), except that the parameters are tied between adjacent nodes.

One problem with PL is that it is hard to apply to models with hidden variables (Parise and Welling 2005). Another more subtle problem is that each node assumes that its neighbors have known values. If node $t \in \text{nbr}(s)$ is a perfect predictor for node s, then s will learn to rely completely on node t, even at the expense of ignoring other potentially useful information, such as its local evidence.

However, experiments in (Parise and Welling 2005; Hoefling and Tibshirani 2009) suggest that PL works as well as exact ML for fully observed Ising models, and of course PL is *much* faster.

19.5.5 Stochastic maximum likelihood

Recall that the gradient of the log-likelihood for a fully observed MRF is given by

$$\nabla_{\boldsymbol{\theta}} \ell(\boldsymbol{\theta}) = \frac{1}{N} \sum_i [\boldsymbol{\phi}(\mathbf{y}_i) - \mathbb{E}[\boldsymbol{\phi}(\mathbf{y})]] \tag{19.51}$$

The gradient for a partially observed MRF is similar. In both cases, we can approximate the model expectations using Monte Carlo sampling. We can combine this with stochastic gradient descent (Section 8.5.2), which takes samples from the empirical distribution. Pseudocode for the resulting method is shown in Algorithm 3.

Algorithm 19.1: Stochastic maximum likelihood for fitting an MRF

```
1  Initialize weights $\boldsymbol{\theta}$ randomly;
2  $k = 0, \eta = 1$ ;
3  for each epoch do
4      for each minibatch of size $B$ do
5          for each sample $s = 1 : S$ do
6              Sample $\mathbf{y}^{s,k} \sim p(\mathbf{y}|\boldsymbol{\theta}_k)$ ;
7          $\hat{E}(\boldsymbol{\phi}(\mathbf{y})) = \frac{1}{S}\sum_{s=1}^{S} \boldsymbol{\phi}(\mathbf{y}^{s,k})$;
8          for each training case $i$ in minibatch do
9              $\mathbf{g}_{ik} = \boldsymbol{\phi}(\mathbf{y}_i) - \hat{E}(\boldsymbol{\phi}(\mathbf{y}))$ ;
10         $\mathbf{g}_k = \frac{1}{B}\sum_{i \in B} \mathbf{g}_{ik}$;
11         $\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \eta \mathbf{g}_k$;
12         $k = k + 1$;
13         Decrease step size $\eta$;
```

Typically we use MCMC to generate the samples. Of course, running MCMC to convergence at each step of the inner loop would be extremely slow. Fortunately, it was shown in (Younes 1989) that we can start the MCMC chain at its previous value, and just take a few steps. In otherwords, we sample $\mathbf{y}^{s,k}$ by initializing the MCMC chain at $\mathbf{y}^{s,k-1}$, and then run for a few iterations. This is valid since $p(\mathbf{y}|\boldsymbol{\theta}^k)$ is likely to be close to $p(\mathbf{y}|\boldsymbol{\theta}^{k-1})$, since we only changed the parameters a small amount. We call this algorithm **stochastic maximum likelihood** or **SML**. (There is a closely related algorithm called persistent contrastive divergence which we discuss in Section 27.7.2.5.)

19.5.6 Feature induction for maxent models *

MRFs require a good set of features. One unsupervised way to learn such features, known as **feature induction**, is to start with a base set of features, and then to continually create new feature combinations out of old ones, greedily adding the best ones to the model. This approach was first proposed in (Pietra et al. 1997; Zhu et al. 1997), and was later extended to the CRF case in (McCallum 2003).

To illustrate the basic idea, we present an example from (Pietra et al. 1997), which described how to build unconditional probabilistic models to represent English spelling. Initially the model has no features, which represents the uniform distribution. The algorithm starts by choosing to add the feature

$$\phi_1(\mathbf{y}) = \sum_t \mathbb{I}(y_t \in \{a, \ldots, z\}) \tag{19.52}$$

which checks if any letter is lower case or not. After the feature is added, the parameters are (re)-fit by maximum likelihood. For this feature, it turns out that $\hat{\theta}_1 = 1.944$, which means that a word with a lowercase letter in any position is about $e^{1.944} \approx 7$ times more likely than the same word without a lowercase letter in that position. Some samples from this model, generated using (annealed) Gibbs sampling (Section 24.2), are shown below.[8]

```
m, r, xevo, ijjiir, b, to, jz, gsr, wq, vf, x, ga, msmGh, pcp, d, oziVlal,
hzagh, yzop, io, advzmxnv, ijv_bolft, x, emx, kayerf, mlj, rawzyb, jp, ag,
ctdnnnbg, wgdw, t, kguv, cy, spxcq, uzflbbf, dxtkkn, cxwx, jpd, ztzh, lv,
zhpkvnu, l^, r, qee, nynrx, atze4n, ik, se, w, lrh, hp+, yrqyka'h, zcngotcnx,
igcump, zjcjs, lqpWiqu, cefmfhc, o, lb, fdcY, tzby, yopxmvk, by, fz,, t, govyccm,
ijyiduwfzo, 6xr, duh, ejv, pk, pjw, l, fl, w
```

The second feature added by the algorithm checks if two adjacent characters are lower case:

$$\phi_2(\mathbf{y}) = \sum_{s \sim t} \mathbb{I}(y_s \in \{a, \ldots, z\}, y_t \in \{a, \ldots, z\}) \tag{19.53}$$

Now the model has the form

$$p(\mathbf{y}) = \frac{1}{Z} \exp(\theta_1 \phi_1(\mathbf{y}) + \theta_2 \phi_2(\mathbf{y})) \tag{19.54}$$

Continuing in this way, the algorithm adds features for the strings `s>` and `ing>`, where $>$ represents the end of word, and for various regular expressions such as `[0-9]`, etc. Some samples from the model with 1000 features, generated using (annealed) Gibbs sampling, are shown below.

```
was, reaser, in, there, to, will, ,, was, by, homes, thing, be, reloverated,
ther, which, conists, at, fores, anditing, with, Mr., proveral, the, ,, ***,
on't, prolling, prothere, ,, mento, at, yaou, 1, chestraing, for, have, to,
intrally, of, qut, ., best, compers, ***, cluseliment, uster, of, is, deveral,
this, thise, of, offect, inatever, thifer, constranded, stater, vill, in, thase,
in, youse, menttering, and, ., of, in, verate, of, to
```

This approach of feature learning can be thought of as a form of graphical model structure learning (Chapter 26), except it is more fine-grained: we add features that are useful, regardless of the resulting graph structure. However, the resulting graphs can become densely connected, which makes inference (and hence parameter estimation) intractable.

19.5.7 Iterative proportional fitting (IPF) *

Consider a pairwise MRF where the potentials are represented as tables, with one parameter per variable setting. We can represent this in log-linear form using

$$\psi_{st}(y_s, y_t) = \exp\left(\boldsymbol{\theta}_{st}^T [\mathbb{I}(y_s = 1, y_t = 1), \ldots, \mathbb{I}(y_s = K, y_t = K)]\right) \tag{19.55}$$

and similarly for $\psi_t(y_t)$. Thus the feature vectors are just indicator functions.

8. We thank John Lafferty for sharing this example.

From Equation 19.43, we have that, at the maximum of the likelihood, the empirical expectation of the features equals the model's expectation:

$$\begin{aligned}\mathbb{E}_{p_{\text{emp}}}\left[\mathbb{I}(y_s = j, y_t = k)\right] &= \mathbb{E}_{p(\cdot|\boldsymbol{\theta})}\left[\mathbb{I}(y_s = j, y_t = k)\right] && (19.56)\\ p_{\text{emp}}(y_s = j, y_t = k) &= p(y_s = j, y_t = k|\boldsymbol{\theta}) && (19.57)\end{aligned}$$

where p_{emp} is the empirical probability:

$$p_{\text{emp}}(y_s = j, y_t = k) = \frac{N_{st,jk}}{N} = \frac{\sum_{n=1}^{N} \mathbb{I}(y_{ns} = j, y_{nt} = k)}{N} \tag{19.58}$$

For a general graph, the condition that must hold at the optimum is

$$p_{\text{emp}}(\mathbf{y}_c) = p(\mathbf{y}_c|\boldsymbol{\theta}) \tag{19.59}$$

For a special family of graphs known as decomposable graphs (defined in Section 20.4.1), one can show that $p(\mathbf{y}_c|\boldsymbol{\theta}) = \psi_c(\mathbf{y}_c)$. However, even if the graph is not decomposable, we can imagine trying to enforce this condition. This suggests an iterative coordinate ascent scheme where at each step we compute

$$\psi_c^{t+1}(\mathbf{y}_c) = \psi_c^t(\mathbf{y}_c) \times \frac{p_{\text{emp}}(\mathbf{y}_c)}{p(\mathbf{y}_c|\psi^t)} \tag{19.60}$$

where the multiplication is elementwise. This is known as **iterative proportional fitting** or **IPF** (Fienberg 1970; Bishop et al. 1975). See Algorithm 7 for the pseudocode.

Algorithm 19.2: Iterative Proportional Fitting algorithm for tabular MRFs

```
1 Initialize $\psi_c = 1$ for $c = 1 : C$;
2 repeat
3     for $c = 1 : C$ do
4         $p_c = p(\mathbf{y}_c|\boldsymbol{\psi})$;
5         $\hat{p}_c = p_{\text{emp}}(\mathbf{y}_c)$;
6         $\psi_c = \psi_c * \frac{\hat{p}_c}{p_c}$;
7 until converged;
```

19.5.7.1 Example

Let us consider a simple example from `http://en.wikipedia.org/wiki/Iterative_proportional_fitting`. We have two binary variables, Y_1 and Y_2, where $Y_{n1} = 1$ if man n is left handed, and $Y_{n1} = 0$ otherwise; similarly, $Y_{n2} = 1$ if woman n is left handed, and $Y_{n2} = 0$ otherwise. We can summarize the data using the following 2×2 contingency table:

	right-handed	left-handed	Total
male	43	9	52
female	44	4	48
Total	87	13	100

Suppose we want to fit a disconnected graphical model containing nodes Y_1 and Y_2 but with no edge between them. That is, we want to find vectors $\boldsymbol{\psi}_1$ and $\boldsymbol{\psi}_2$ such that $\mathbf{M} \triangleq \boldsymbol{\psi}_1 \boldsymbol{\psi}_2^T \approx \mathbf{C}$, where $\mathbf{M}$ are the model's expected counts, and $\mathbf{C}$ are the empirical counts. By moment matching, we find that the row and column sums of the model must exactly match the row and column sums of the data. One possible solution is to use $\boldsymbol{\psi}_1 = [0.5200, 0.4800]$ and $\boldsymbol{\psi}_2 = [87, 13]$. Below we show the model's predictions, $\mathbf{M} = \boldsymbol{\psi}_1 \boldsymbol{\psi}_2^T$.

	right-handed	left-handed	Total
male	45.24	6.76	52
female	41.76	6.24	48
Total	87	13	100

It is easy to see that this matches the required constraints. See `IPFdemo2x2` for some Matlab code that computes these numbers. This method is easily to generalized to arbitrary graphs.

19.5.7.2 Speed of IPF

IPF is a fixed point algorithm for enforcing the moment matching constraints and is guaranteed to converge to the global optimum (Bishop et al. 1975). The number of iterations depends on the form of the model. If the graph is decomposable, then IPF converges in a single iteration, but in general, IPF may require many iterations.

It is clear that the dominant cost of IPF is computing the required marginals under the model. Efficient methods, such as the junction tree algorithm (Section 20.4), can be used, resulting in something called **efficient IPF** (Jirousek and Preucil 1995).

Nevertheless, coordinate descent can be slow. An alternative method is to update all the parameters at once, by simply following the gradient of the likelihood. This gradient approach has the further significant advantage that it works for models in which the clique potentials may not be fully parameterized, i.e., the features may not consist of all possible indicators for each clique, but instead can be arbitrary. Although it is possible to adapt IPF to this setting of general features, resulting in a method known as **iterative scaling**, in practice the gradient method is much faster (Malouf 2002; Minka 2003).

19.5.7.3 Generalizations of IPF

We can use IPF to fit Gaussian graphical models: instead of working with empirical counts, we work with empirical means and covariances (Speed and Kiiveri 1986). It is also possible to create a Bayesian IPF algorithm for sampling from the posterior of the model's parameters (see e.g., (Dobra and Massam 2010)).

19.5.7.4 IPF for decomposable graphical models

There is a special family of undirected graphical models known as decomposable graphical models. This is formally defined in Section 20.4.1, but the basic idea is that it contains graphs which are "tree-like". Such graphs can be represented by UGMs or DGMs without any loss of information.

In the case of decomposable graphical models, IPF converges in one iteration. In fact, the

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).

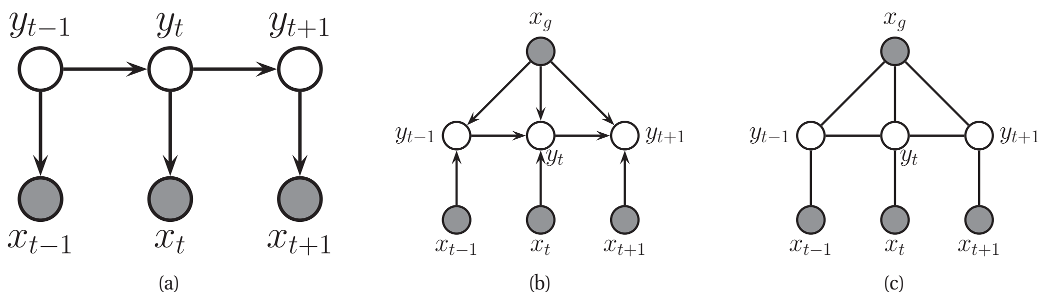


Figure 19.14 Various models for sequential data. (a) A generative directed HMM. (b) A discriminative directed MEMM. (c) A discriminative undirected CRF.

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,

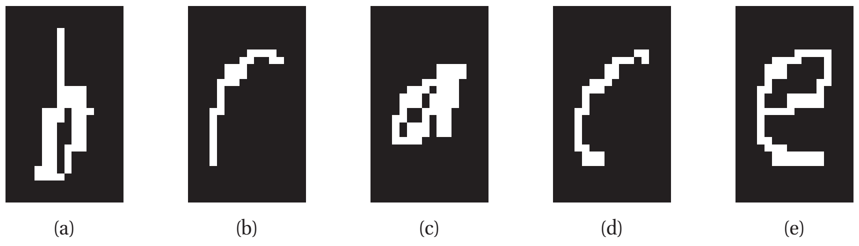


Figure 19.15 Example of handwritten letter recognition. In the word 'brace', the 'r' and the 'c' look very similar, but can be disambiguated using context. Source: (Taskar et al. 2003) . Used with kind permission of Ben Taskar.

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.

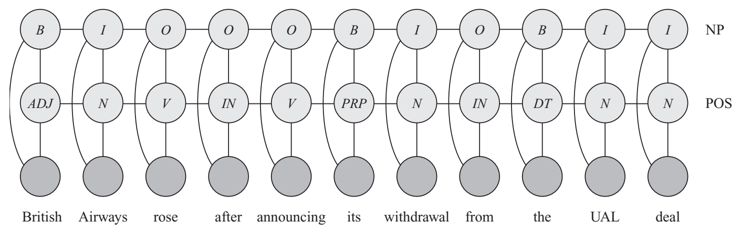
The label bias problem in MEMMs occurs because directed models are **locally normalized**, meaning each CPD sums to 1. By contrast, MRFs and CRFs are **globally normalized**, which means that local factors do not need to sum to 1, since the partition function Z, which sums over all joint configurations, 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,



KEY

B	Begin noun phrase	*V*	Verb
I	Within noun phrase	*IN*	Preposition
O	Not a noun phrase	*PRP*	Possesive pronoun
N	Noun	*DT*	Determiner (e.g., a, an, the)
ADJ	Adjective		

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**.

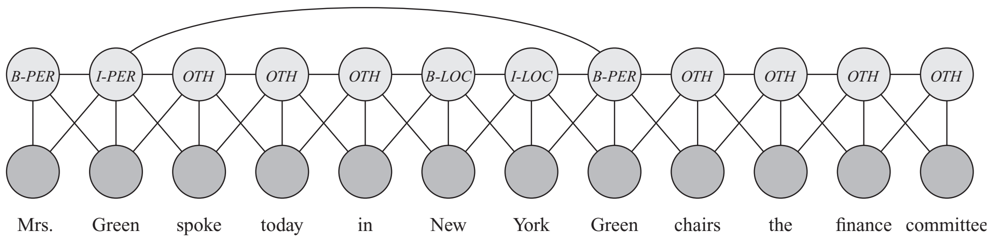
In more detail, we tag each word in the sentence with B (meaning beginning of a new NP), I (meaning inside a NP), or O (meaning outside an NP). This is called **BIO** notation. For example, in the following sentence, the NPs are marked with brackets:

```
    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



KEY			
B-PER	Begin person name	*I-LOC*	Within location name
I-PER	Within person name	*OTH*	Not an entitiy
B-LOC	Begin location name		

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.

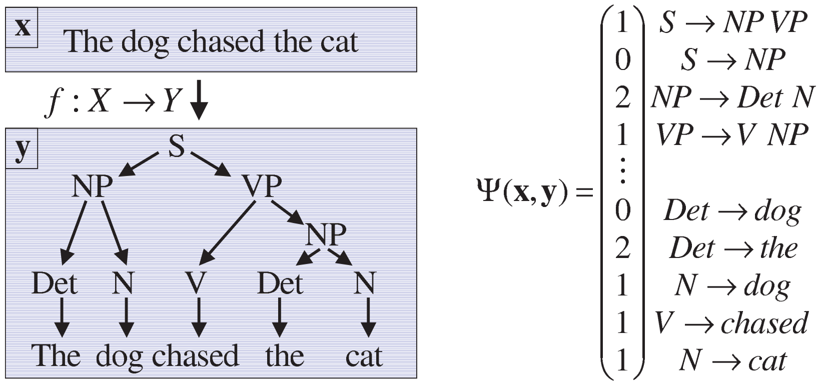


Figure 19.18 Illustration of a simple parse tree based on a context free grammar in Chomsky normal form. The feature vector $\boldsymbol{\phi}(\mathbf{x}, \mathbf{y}) = \Psi(\mathbf{x}, \mathbf{y})$ counts the number of times each production rule was used. Source: Figure 5.2 of (Altun et al. 2006) .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

19.6.2.4 Natural language parsing

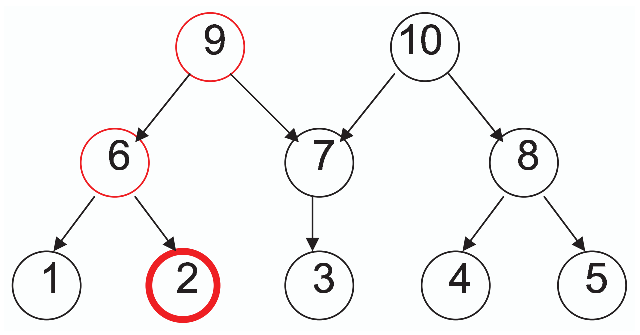
A generalization of chain-structured models for language is to use probabilistic grammars. In particular, a probabilistic **context free grammar** or **PCFG** is a set of re-write or production rules of the form $\sigma \to \sigma'\sigma''$ or $\sigma \to x$, where $\sigma, \sigma', \sigma'' \in \Sigma$ 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 $\mathbf{x} = x_1 \dots x_T$ 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.



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

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

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 $\mathbf{y}^* = \text{argmin}\, E(\mathbf{x}, \mathbf{y}|\mathbf{w})$. 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**.

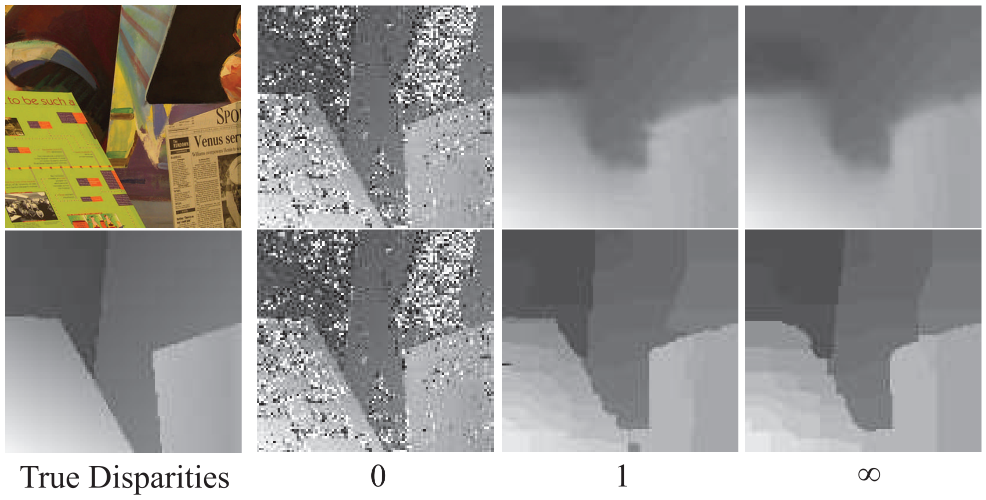


Figure 19.20 Illustration of belief propagation for stereo depth estimation. Left column: image and true disparities. Remaining columns: initial estimate, estimate after 1 iteration, and estimate at convergence. Top row: Gaussian edge potentials. Bottom row: robust edge potentials. Source: Figure 4 of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

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

However, at test time, we pick the label so as to minimize the posterior expected loss (defined in Section 5.7):

$$\hat{\mathbf{y}}(\mathbf{x}|\mathbf{w}) = \underset{\hat{\mathbf{y}}}{\operatorname{argmin}} \sum_{\mathbf{y}} L(\hat{\mathbf{y}}, \mathbf{y}) p(\mathbf{y}|\mathbf{x}, \mathbf{w}) \tag{19.79}$$

where $L(\mathbf{y}^*, \hat{\mathbf{y}})$ is the loss we incur when we estimate $\hat{\mathbf{y}}$ but the truth is $\mathbf{y}^*$. It therefore seems reasonable to take the loss function into account when performing parameter estimation.[10] So, following (Yuille and He 2011), let us instead minimized the posterior expected loss on the training set:

$$R_{EL}(\mathbf{w}) \triangleq -\log p(\mathbf{w}) + \sum_{i=1}^{N} \log \left[\sum_{\mathbf{y}} L(\mathbf{y}_i, \mathbf{y}) p(\mathbf{y}|\mathbf{x}_i, \mathbf{w}) \right] \tag{19.80}$$

In the special case of 0-1 loss, $L(\mathbf{y}_i, \mathbf{y}) = 1 - \delta_{\mathbf{y}, \mathbf{y}_i}$, this reduces to R_{MAP}.

We will assume that we can write our model in the following form:

$$p(\mathbf{y}|\mathbf{x}, \mathbf{w}) = \frac{\exp(\mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}, \mathbf{y}))}{Z(\mathbf{x}, \mathbf{w})} \tag{19.81}$$

$$p(\mathbf{w}) = \frac{\exp(-E(\mathbf{w}))}{Z} \tag{19.82}$$

where $Z(\mathbf{x}, \mathbf{w}) = \sum_{\mathbf{y}} \exp(\mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}, \mathbf{y}))$. Also, let us define $L(\mathbf{y}_i, \mathbf{y}) = \exp \tilde{L}(\mathbf{y}_i, \mathbf{y})$. With this, we can rewrite our objective as follows:

$$R_{EL}(\mathbf{w}) = -\log p(\mathbf{w}) + \sum_i \log \left[\sum_{\mathbf{y}} \exp \tilde{L}(\mathbf{y}_i, \mathbf{y}) \frac{\exp(\mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}, \mathbf{y}))}{Z(\mathbf{x}, \mathbf{w})} \right] \tag{19.83}$$

$$= E(\mathbf{w}) + \sum_i -\log Z(\mathbf{x}_i, \mathbf{w}) + \log \sum_{\mathbf{y}} \exp \left(\tilde{L}(vy_i, \mathbf{y}) + \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}_i, \mathbf{y}) \right) \tag{19.84}$$

We will now consider various bounds in order to simplify this objective. First note that for any function $f(\mathbf{y})$ we have

$$\max_{\mathbf{y} \in \mathcal{Y}} f(\mathbf{y}) \leq \log \sum_{\mathbf{y} \in \mathcal{Y}} \exp[f(\mathbf{y})] \leq \log \left[|\mathcal{Y}| \exp \left(\max_{\mathbf{y}} f(\mathbf{y}) \right) \right] = \log |\mathcal{Y}| + \max_{\mathbf{y}} f(\mathbf{y}) \tag{19.85}$$

For example, suppose $\mathcal{Y} = \{0, 1, 2\}$ and $f(y) = y$. Then we have

$$2 = \log[\exp(2)] \leq \log[\exp(0) + \exp(1) + \exp(2)] \leq \log[3 \times \exp(2)] = \log(3) + 2 \tag{19.86}$$

We can ignore the $\log |\mathcal{Y}|$ term, which is independent of $\mathbf{y}$, and treat $\max_{\mathbf{y} \in \mathcal{Y}} f(\mathbf{y})$ as both a lower and upper bound. Hence we see that

$$R_{EL}(\mathbf{w}) \sim E(\mathbf{w}) + \sum_{i=1}^{N} \left[\max_{\mathbf{y}} \left\{ \tilde{L}(\mathbf{y}_i, \mathbf{y}) + \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}_i, \mathbf{y}) \right\} - \max_{\mathbf{y}} \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}_i, \mathbf{y}) \right] \tag{19.87}$$

10. Note that this violates the fundamental Bayesian distinction between inference and decision making. However, performing these tasks separately will only result in an optimal decision if we can compute the exact posterior. In most cases, this is intractable, so we need to perform **loss-calibrated inference** (Lacoste-Julien et al. 2011). In this section, we just perform loss-calibrated MAP parameter estimation, which is computationally simpler. (See also (Stoyanov et al. 2011).)

where $x \sim y$ means $c_1 + x \leq y + c_2$ for some constants c_1, c_2. Unfortunately, this objective is not convex in $\mathbf{w}$. However, we can devise a convex upper bound by exploiting the following looser lower bound on the log-sum-exp function:

$$f(\mathbf{y}') \leq \log \sum_{\mathbf{y}} \exp[f(\mathbf{y})] \tag{19.88}$$

for any $\mathbf{y}' \in \mathcal{Y}$. Applying this equation to our earlier example, for $f(y) = y$ and $y' = 1$, we get $1 = \log[\exp(1)] \leq \log[\exp(0) + \exp(1) + \exp(2)]$. And applying this bound to R_{EL} we get

$$R_{EL}(\mathbf{w}) \quad \leq \quad E(\mathbf{w}) + \sum_{i=1}^{N} \left[\max_{\mathbf{y}} \left\{ \tilde{L}(\mathbf{y}_i, \mathbf{y}) + \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}_i, \mathbf{y}) \right\} - \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}_i, \mathbf{y}_i) \right] \tag{19.89}$$

If we set $E(\mathbf{w}) = -\frac{1}{2C}||\mathbf{w}||_2^2$ (corresponding to a spherical Gaussian prior), we get

$$R_{SSVM}(\mathbf{w}) \quad \triangleq \quad \frac{1}{2}||\mathbf{w}||^2 + C \sum_{i=1}^{N} \left[\max_{\mathbf{y}} \left\{ \tilde{L}(\mathbf{y}_i, \mathbf{y}) + \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}_i, \mathbf{y}) \right\} - \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}_i, \mathbf{y}_i) \right] \tag{19.90}$$

This is the same objective as used in the SSVM approach of (Tsochantaridis et al. 2005).

In the special case that $\mathcal{Y} = \{-1, +1\}$ $L(y^*, y) = 1 - \delta_{y,y^*}$, and $\boldsymbol{\phi}(\mathbf{x}, y) = \frac{1}{2} y \mathbf{x}$, this criterion reduces to the following (by considering the two cases that $y = y_i$ and $y \neq y_i$):

$$R_{SVM}(\mathbf{w}) \quad \triangleq \quad \frac{1}{2}||\mathbf{w}||^2 + C \sum_{i=1}^{N} \left[\max\{0, 1 - y_i \mathbf{w}^T \mathbf{x}_i\} \right] \tag{19.91}$$

which is the standard binary SVM objective (see Equation 14.57).

So we see that the SSVM criterion can be seen as optimizing an upper bound on the Bayesian objective, a result first shown in (Yuille and He 2011). This bound will be tight (and hence the approximation will be a good one) when $||\mathbf{w}||$ is large, since in that case, $p(\mathbf{y}|\mathbf{x}, \mathbf{w})$ will concentrate its mass on $\operatorname{argmax}_{\mathbf{y}} p(\mathbf{y}|\mathbf{x}, \mathbf{w})$. Unfortunately, a large $||\mathbf{w}||$ corresponds to a model that is likely to overfit, so it is unlikely that we will be working in this regime (because we will tune the strength of the regularizer to avoid this situation). An alternative justification for the SVM criterion is that it focusses effort on fitting parameters that affect the decision boundary. This is a better use of computational resources than fitting the full distribution, especially when the model is wrong.

19.7.2 SSVMs: a non-probabilistic view

We now present SSVMs in a more traditional (non-probabilistic) way, following (Tsochantaridis et al. 2005). The resulting objective will be the same as the one above. However, this derivation will set the stage for the algorithms we discuss below.

Let $f(\mathbf{x}; \mathbf{w}) = \operatorname{argmax}_{\mathbf{y} \in \mathcal{Y}} \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}, \mathbf{y})$ be the prediction function. We can obtain zero loss on the training set using this predictor if

$$\forall i. \max_{\mathbf{y} \in \mathcal{Y} \setminus \mathbf{y}_i} \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}_i, \mathbf{y}) \leq \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}_i, \mathbf{y}_i) \tag{19.92}$$

Each one of these nonlinear inequalities can be equivalently replaced by $|\mathcal{Y}| - 1$ linear inequalities, resulting in a total of $N|\mathcal{Y}| - N$ linear constraints of the following form:

$$\forall i . \forall \mathbf{y} \in \mathcal{Y} \setminus \mathbf{y}_i . \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}_i, \mathbf{y}_i) - \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}_i, \mathbf{y}) \geq 0 \tag{19.93}$$

For brevity, we introduce the notation

$$\boldsymbol{\delta}_i(\mathbf{y}) \triangleq \boldsymbol{\phi}(\mathbf{x}_i, \mathbf{y}_i) - \boldsymbol{\phi}(\mathbf{x}_i, \mathbf{y}) \tag{19.94}$$

so we can rewrite these constraints as $\mathbf{w}^T \boldsymbol{\delta}_i(\mathbf{y}) \geq 0$.

If we can achieve zero loss, there will typically be multiple solution vectors $\mathbf{w}$. We pick the one that maximizes the margin, defined as

$$\gamma \triangleq \min_i f(\mathbf{x}, \mathbf{y}_i; \mathbf{w}) - \max_{\mathbf{y}' \in \mathcal{Y} \setminus \mathbf{y}} f(\mathbf{x}, \mathbf{y}'; \mathbf{w}) \tag{19.95}$$

Since the margin can be made arbitrarily large by rescaling $\mathbf{w}$, we fix its norm to be 1, resulting in the optimization problem

$$\max_{\gamma, \mathbf{w}: ||\mathbf{w}||=1} \quad \text{s.t.} \quad \forall i . \forall \mathbf{y} \in \mathcal{Y} \setminus \mathbf{y}_i . \;\; \mathbf{w}^T \boldsymbol{\delta}_i(\mathbf{y}) \geq \gamma \tag{19.96}$$

Equivalently, we can write

$$\min_{\mathbf{w}} \frac{1}{2} ||\mathbf{w}||^2 \quad \text{s.t.} \quad \forall i . \forall \mathbf{y} \in \mathcal{Y} \setminus \mathbf{y}_i . \;\; \mathbf{w}^T \boldsymbol{\delta}_i(\mathbf{y}) \geq 1 \tag{19.97}$$

To allow for the case where zero loss cannot be achieved (equivalent to the data being inseparable in the case of binary classification), we relax the constraints by introducing slack terms ξ_i, one per data case. This yields

$$\min_{\mathbf{w}, \boldsymbol{\xi}} \frac{1}{2} ||\mathbf{w}||^2 + C \sum_{i=1}^N \xi_i \quad \text{s.t.} \quad \forall i . \forall \mathbf{y} \in \mathcal{Y} \setminus \mathbf{y}_i . \;\; \mathbf{w}^T \boldsymbol{\delta}_i(\mathbf{y}) \geq 1 - \xi_i, \xi_i \geq 0 \tag{19.98}$$

In the case of structured outputs, we don't want to treat all constraint violations equally. For example, in a segmentation problem, getting one position wrong should be punished less than getting many positions wrong. One way to achieve this is to divide the slack variable by the size of the loss (this is called **slack re-scaling**). This yields

$$\min_{\mathbf{w}, \boldsymbol{\xi}} \frac{1}{2} ||\mathbf{w}||^2 + C \sum_{i=1}^N \xi_i \quad \text{s.t.} \quad \forall i . \forall \mathbf{y} \in \mathcal{Y} \setminus \mathbf{y}_i . \;\; \mathbf{w}^T \boldsymbol{\delta}_i(\mathbf{y}) \geq 1 - \frac{\xi_i}{L(\mathbf{y}_i, \mathbf{y})}, \xi_i \geq 0 \tag{19.99}$$

Alternatively, we can define the margin to be proportional to the loss (this is called **margin re-rescaling)**. This yields

$$\min_{\mathbf{w}, \boldsymbol{\xi}} \frac{1}{2} ||\mathbf{w}||^2 + C \sum_{i=1}^N \xi_i \quad \text{s.t.} \quad \forall i . \forall \mathbf{y} \in \mathcal{Y} \setminus \mathbf{y}_i . \;\; \mathbf{w}^T \boldsymbol{\delta}_i(\mathbf{y}) \geq L(\mathbf{y}_i, \mathbf{y}) - \xi_i, \;\; \xi_i \geq 0 \tag{19.100}$$

(In fact, we can write $\forall \mathbf{y} \in \mathcal{Y}$ instead of $\forall \mathbf{y} \in \mathcal{Y} \setminus \mathbf{y}_i$, since if $\mathbf{y} = \mathbf{y}_i$, then $\mathbf{w}^T \boldsymbol{\delta}_i(\mathbf{y}) = 0$ and $\xi_i = 0$. By using the simpler notation, which doesn't exclude $\mathbf{y}_i$, we add an extra but redundant constraint.) This latter approach is used in M3nets.

For future reference, note that we can solve for the ξ_i^* terms as follows:

$$\xi_i^*(\mathbf{w}) = \max\{0, \max_{\mathbf{y}}(L(\mathbf{y}_i, \mathbf{y}) - \mathbf{w}^T \boldsymbol{\delta}_i))\} = \max_{\mathbf{y}}(L(\mathbf{y}_i, \mathbf{y}) - \mathbf{w}^T \boldsymbol{\delta}_i)) \tag{19.101}$$

Substituting in, and dropping the constraints, we get the following equivalent problem:

$$\min_{\mathbf{w}} \frac{1}{2}||\mathbf{w}||^2 + C \sum_i \max_{\mathbf{y}} \left\{ L(\mathbf{y}_i, \mathbf{y}) + \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}_i, \mathbf{y}) \right\} - \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}_i, \mathbf{y}_i) \tag{19.102}$$

19.7.2.1 Empirical risk minimization

Let us pause and consider whether the above objective is reasonable. Recall that in the frequentist approach to machine learning (Section 6.5), the goal is to minimize the regularized empirical risk, defined by

$$\mathcal{R}(\mathbf{w}) + \frac{C}{N} \sum_{i=1}^{N} L(\mathbf{y}_i, f(\mathbf{x}_i, \mathbf{w})) \tag{19.103}$$

where $\mathcal{R}(\mathbf{w})$ is the regularizer, and $f(\mathbf{x}_i, \mathbf{w}) = \text{argmax}_{\mathbf{y}}\, \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}_i, \mathbf{y}) = \hat{\mathbf{y}}_i$ is the prediction. Since this objective is hard to optimize, because the loss is not differentiable, we will construct a convex upper bound instead.

We can show that

$$\mathcal{R}(\mathbf{w}) + \frac{C}{N} \sum_i \max_{\mathbf{y}}(L(\mathbf{y}_i, \mathbf{y}) - \mathbf{w}^T \boldsymbol{\delta}_i)) \tag{19.104}$$

is such a convex upper bound. To see this, note that

$$\begin{aligned} L(\mathbf{y}_i, f(\mathbf{x}_i, \mathbf{w})) &\leq L(\mathbf{y}_i, f(\mathbf{x}_i, \mathbf{w})) - \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}_i, \mathbf{y}_i) + \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}_i, \hat{\mathbf{y}}_i) & (19.105) \\ &\leq \max_{\mathbf{y}} L(\mathbf{y}_i, \mathbf{y}) - \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}_i, \mathbf{y}_i) + \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}_i, \mathbf{y}) & (19.106) \end{aligned}$$

Using this bound and $\mathcal{R}(\mathbf{w}) = \frac{1}{2}||\mathbf{w}||^2$ yields Equation 19.102.

19.7.2.2 Computational issues

Although the above objectives are simple quadratic programs (QP), they have $O(N|\mathcal{Y}|)$ constraints. This is intractable, since $\mathcal{Y}$ is usually exponentially large. In the case of the margin rescaling formulation, it is possible to reduce the exponential number of constraints to a polynomial number, provided the loss function and the feature vector decompose according to a graphical model. This is the approach used in M3nets (Taskar et al. 2003).

An alternative approach is to work directly with the exponentially sized QP. This allows for the use of more general loss functions. There are several possible methods to make this feasible. One is to use cutting plane methods. Another is to use stochastic subgradient methods. We discuss both of these below.

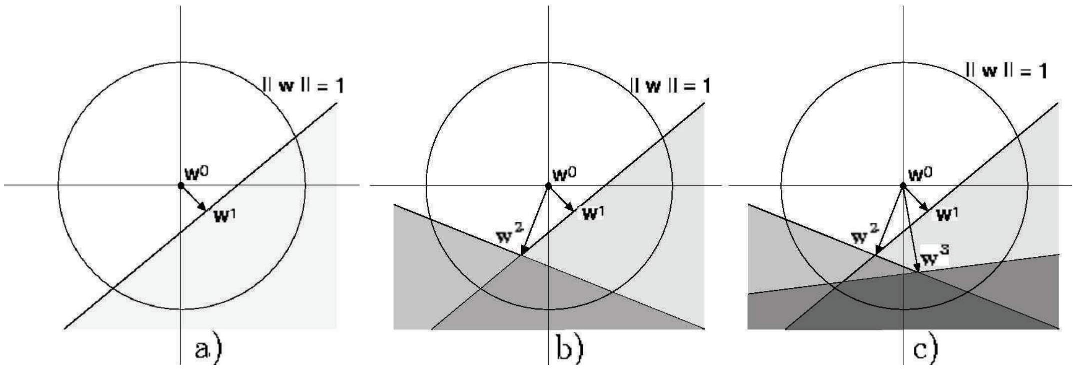


Figure 19.21 Illustration of the cutting plane algorithm in 2d. We start with the estimate $\mathbf{w} = \mathbf{w}_0 = \mathbf{0}$. (a) We add the first constraint; the shaded region is the new feasible set. The new minimum norm solution is $\mathbf{w}_1$. (b) We add another constraint; the dark shaded region is the new feasible set. (c) We add a third constraint. Source: Figure 5.3 of (Altun et al. 2006) . Used with kind permission of Yasemin Altun.

19.7.3 Cutting plane methods for fitting SSVMs

In this section, we discuss an efficient algorithm for fitting SSVMs due to (Joachims et al. 2009). This method can handle general loss functions, and is implemented in the popular **SVMstruct** package[11]. The method is based on the **cutting plane** method from convex optimization (Kelley 1960).

The basic idea is as follows. We start with an initial guess $\mathbf{w}$ and no constraints. At each iteration, we then do the following: for each example i, we find the "most violated" constraint involving $\mathbf{x}_i$ and $\hat{\mathbf{y}}_i$. If the loss-augmented margin violation exceeds the current value of ξ_i by more than ϵ, we add $\hat{\mathbf{y}}_i$ to the working set of constraints for this training case, $\mathcal{W}_i$, and then solve the resulting new QP to find the new $\mathbf{w}, \boldsymbol{\xi}$. See Figure 19.21 for a sketch, and Algorithm 11 for the pseudo code. (Since at each step we only add one new constraint, we can warm-start the QP solver.) We can can easily modify the algorithm to optimize the slack rescaling version by replacing the expression $L(\mathbf{y}_i, \mathbf{y}) - \mathbf{w}^T \boldsymbol{\delta}_i(\hat{\mathbf{y}}_i)$ with $L(\mathbf{y}_i, \mathbf{y})(1 - \mathbf{w}^T \boldsymbol{\delta}_i(\hat{\mathbf{y}}_i))$.

The key to the efficiency of this method is that only polynomially many constraints need to be added, and as soon as they are, the exponential number of other constraints are guaranteed to also be satisfied to within a tolerance of ϵ (see (Tsochantaridis et al. 2005) for the proof).

19.7.3.1 Loss-augmented decoding

The other key to efficiency is the ability to find the most violated constraint in line 5 of the algorithm, i.e., to compute

$$\underset{\mathbf{y} \in \mathcal{Y}}{\operatorname{argmax}}\, L(\mathbf{y}_i, \mathbf{y}) - \mathbf{w}^T \boldsymbol{\delta}_i(\mathbf{y}) = \underset{\mathbf{y} \in \mathcal{Y}}{\operatorname{argmax}}\, L(\mathbf{y}_i, \mathbf{y}) + \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}_i, \mathbf{y}) \tag{19.107}$$

11. `http://svmlight.joachims.org/svm_struct.html`

Algorithm 19.3: Cutting plane algorithm for SSVMs (margin rescaling, N-slack version)

```
1  Input $\mathcal{D} = \{(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_N, \mathbf{y}_n)\}$, $C$, $\epsilon$ ;
2  $\mathcal{W}_i = \emptyset$, $\xi_i = 0$ for $i = 1 : N$ ;
3  repeat
4      for $i = 1 : N$ do
5          $\hat{\mathbf{y}}_i = \text{argmax}_{\hat{\mathbf{y}}_i \in \mathcal{Y}} L(\mathbf{y}_i, \mathbf{y}) - \mathbf{w}^T \boldsymbol{\delta}_i(\hat{\mathbf{y}}_i)$ ;
6          if $L(\mathbf{y}_i, \mathbf{y}) - \mathbf{w}^T \boldsymbol{\delta}_i(\hat{\mathbf{y}}_i) > \xi_i + \epsilon$ then
7              $\mathcal{W}_i = \mathcal{W}_i \cup \{\hat{\mathbf{y}}_i\}$ ;
8              $(\mathbf{w}, \boldsymbol{\xi}) = \text{argmin}_{\mathbf{w}, \boldsymbol{\xi} \geq \mathbf{0}} \frac{1}{2}||\mathbf{w}||_2^2 + C \sum_{i=1}^N \xi_i$ ;
9                  s.t. $\forall i = 1 : N, \forall \mathbf{y}' \in \mathcal{W}_i : \mathbf{w}^T \boldsymbol{\delta}_i(\hat{\mathbf{y}}_i) \geq L(\mathbf{y}_i, \mathbf{y}') - \xi_i$ ;
10 until no $\mathcal{W}_i$ has changed;
11 Return $(\mathbf{w}, \boldsymbol{\xi})$
```

We call this process **loss-augmented decoding**. (In (Joachims et al. 2009), this procedure is called the **separation oracle**.) If the loss function has an additive decomposition of the same form as the features, then we can fold the loss into the weight vector, i.e., we can find a new set of parameters $\mathbf{w}'$ such that $(\mathbf{w}')^T \boldsymbol{\delta}_i(\mathbf{y}) = \mathbf{w}^T \boldsymbol{\delta}_i(\mathbf{y})$. We can then use a standard decoding algorithm, such as Viterbi, on the model $p(\mathbf{y}|\mathbf{x}, \mathbf{w}')$.

In the special case of 0-1 loss, the optimum will either be the best solution, $\text{argmax}_{\mathbf{y}} \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}_i, \mathbf{y})$, with a value of of $0 - \mathbf{w}^T \delta_i(\hat{\mathbf{y}})$, or it will be the second best solution, i.e.,

$$\tilde{\mathbf{y}} = \underset{\mathbf{y} \neq \hat{\mathbf{y}}}{\text{argmax}}\, \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}_i, \mathbf{y}) \tag{19.108}$$

which achieves an overall value of $1 - \mathbf{w}^T \delta_i(\tilde{\mathbf{y}})$. For chain structured CRFs, we can use the Viterbi algorithm to do decoding; the second best path will differ from the best path in a single position, which can be obtained by changing the variable whose max marginal is closest to its decision boundary to its second best value. We can generalize this (with a bit more work) to find the N-best list (Schwarz and Chow 1990; Nilsson and Goldberger 2001).

For Hamming loss, $L(\mathbf{y}^*, \mathbf{y}) = \sum_t \mathbb{I}(y_t^* \neq y_t)$, and for the F1 score (defined in Section 5.7.2.3), we can devise a dynamic programming algorithm to compute Equation 19.107. See (Altun et al. 2006) for details. Other models and loss function combinations will require different methods.

19.7.3.2 A linear time algorithm

Although the above algorithm takes polynomial time, we can do better, and devise an algorithm that runs in *linear* time, assuming we use a linear kernel (i.e., we work with the original features $\boldsymbol{\phi}(\mathbf{x}, \mathbf{y})$ and do not apply the kernel trick). The basic idea, as explained in (Joachims et al. 2009), is to have a single slack variable, ξ, instead of N, but to use $|\mathcal{Y}|^N$ constraints, instead of just $N|\mathcal{Y}|$. Specifically, we optimize the following (assuming the margin rescaling formulation):

$$\min_{\mathbf{w},\xi\geq 0} \quad \frac{1}{2}||\mathbf{w}||_2^2 + C\xi$$

$$\text{s.t.} \quad \forall(\overline{\mathbf{y}}_1,\ldots,\overline{\mathbf{y}}_N)\in\mathcal{Y}^N : \frac{1}{N}\mathbf{w}^T\sum_{i=1}^N \boldsymbol{\delta}_i(\overline{\mathbf{y}}_i) \geq \frac{1}{N}\sum_{i=1}^N L(\mathbf{y}_i,\overline{\mathbf{y}}_i) - \xi \quad (19.109)$$

Compare this to the original version, which was

$$\min_{\mathbf{w},\boldsymbol{\xi}\geq 0} \frac{1}{2}||\mathbf{w}||_2^2 + \frac{C}{N}\xi \quad \text{s.t.} \quad \forall i=1:N, \forall \mathbf{y}\in\mathcal{Y} : \mathbf{w}^T\boldsymbol{\delta}_i(\mathbf{y}) \geq L(\mathbf{y}_i,\overline{\mathbf{y}}_i) - \xi_i \quad (19.110)$$

One can show that any solution $\mathbf{w}^*$ of Equation 19.109 is also a solution of Equation 19.110 and vice versa, with $\xi^* = \frac{1}{N}\xi_i^*$.

Algorithm 19.4: Cutting plane algorithm for SSVMs (margin rescaling, 1-slack version)

```
1  Input $\mathcal{D} = \{(\mathbf{x}_1,\mathbf{y}_1),\ldots,(\mathbf{x}_N,\mathbf{y}_n)\}$, $C$, $\epsilon$ ;
2  $\mathcal{W} = \emptyset$;
3  repeat
4      $(\mathbf{w},\xi) = \text{argmin}_{\mathbf{w},\xi\geq 0}\frac{1}{2}||\mathbf{w}||_2^2 + C\sum_{i=1}^N \xi$ ;
5              s.t.  $\forall(\overline{\mathbf{y}}_1,\ldots,\overline{\mathbf{y}}_N)\in\mathcal{W} : \frac{1}{N}\mathbf{w}^T\sum_{i=1}^N\boldsymbol{\delta}_i(\overline{\mathbf{y}}_i) \geq \frac{1}{N}\sum_{i=1}^N L(\mathbf{y}_i,\overline{\mathbf{y}}_i) - \xi$;
6      for $i = 1:N$ do
7          $\hat{\mathbf{y}}_i = \text{argmax}_{\hat{\mathbf{y}}_i\in\mathcal{Y}} L(\mathbf{y}_i,\hat{\mathbf{y}}_i) + \mathbf{w}^T\boldsymbol{\phi}(\mathbf{x}_i,\hat{\mathbf{y}}_i)$
8      $\mathcal{W} = \mathcal{W}\cup\{(\hat{\mathbf{y}}_1,\ldots,\hat{\mathbf{y}}_N)\}$;
9  until $\frac{1}{N}\sum_{i=1}^N L(\mathbf{y}_i,\hat{\mathbf{y}}_i) - \frac{1}{N}\mathbf{w}^T\sum_{i=1}^N\boldsymbol{\delta}_i(\hat{\mathbf{y}}_i) \leq \xi + \epsilon$;
10 Return $(\mathbf{w},\xi)$
```

We can optimize Equation 19.109 using the cutting plane algorithm in Algorithm 10. (This is what is implemented in SVMstruct.) The inner QP in line 4 can be solved in $O(N)$ time using the method of (Joachims 2006). In line 7 we make N calls to the loss-augmented decoder. Finally, it can be shown that the number of iterations is a constant independent on N. Thus the overall running time is linear.

19.7.4 Online algorithms for fitting SSVMs

Although the cutting plane algorithm can be made to run in time linear in the number of data points, that can still be slow if we have a large dataset. In such cases, it is preferable to use online learning. We briefly mention a few possible algorithms below.

19.7.4.1 The structured perceptron algorithm

A very simple algorithm for fitting SSVMs is the **structured perceptron algorithm** (Collins 2002). This method is an extension of the regular perceptron algorithm of Section 8.5.4. At each step, we compute $\hat{\mathbf{y}} = \operatorname{argmax} p(\mathbf{y}|\mathbf{x})$ (e.g., using the Viterbi algorithm) for the current training sample $\mathbf{x}$. If $\hat{\mathbf{y}} = \mathbf{y}$, we do nothing, otherwise we update the weight vector using

$$\mathbf{w}_{k+1} = \mathbf{w}_k + \boldsymbol{\phi}(\mathbf{y}, \mathbf{x}) - \boldsymbol{\phi}(\hat{\mathbf{y}}, \mathbf{x}) \tag{19.111}$$

To get good performance, it is necessary to average the parameters over the last few updates (see Section 8.5.2 for details), rather than using the most recent value.

19.7.4.2 Stochastic subgradient descent

The disadvantage of the structured perceptron algorithm is that it implicitly assumes 0-1 loss, and it does not enforce any kind of margin. An alternative approach is to perform stochastic subgradient descent. A specific instance of this the **Pegasos** algorithm (Shalev-Shwartz et al. 2007), which stands for "primal estimated sub-gradient solver for SVM". Pegasos was designed for binary SVMs, but can be extended to SSVMS.

Let us start by considering the objective function:

$$f(\mathbf{w}) = \sum_{i=1}^{N} \max_{\hat{\mathbf{y}}_i} \left[L(\mathbf{y}_i, \hat{\mathbf{y}}_i) + \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}_i, \hat{\mathbf{y}}_i) \right] - \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}_i, \mathbf{y}_i) + \lambda ||\mathbf{w}||^2 \tag{19.112}$$

Letting $\hat{\mathbf{y}}_i$ be the argmax of this max. Then the subgradient of this objective function is

$$g(\mathbf{w}) = \sum_{i=1}^{N} \boldsymbol{\phi}(\mathbf{x}_i, \hat{\mathbf{y}}_i) - \boldsymbol{\phi}(\mathbf{x}_i, \mathbf{y}_i) + 2\lambda \mathbf{w} \tag{19.113}$$

In stochastic subgradient descent, we approximate this gradient with a single term, i, and then perform an update:

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \eta_k g_i(\mathbf{w}_k) = \mathbf{w}_k - \eta_k [\boldsymbol{\phi}(\mathbf{x}_i, \hat{\mathbf{y}}_i) - \boldsymbol{\phi}(\mathbf{x}_i, \mathbf{y}_i) + (2/N)\lambda \mathbf{w}] \tag{19.114}$$

where η_k is the step size parameter, which should satisfy the Robbins-Monro conditions (Section 8.5.2.1). (Notice that the perceptron algorithm is just a special case where $\lambda = 0$ and $\eta_k = 1$.) To ensure that $\mathbf{w}$ has unit norm, we can project it onto the ℓ_2 ball after each update.

19.7.5 Latent structural SVMs

In many applications of interest, we have latent or hidden variables $\mathbf{h}$. For example, in object detections problems, we may be told that the image contains an object, so $y = 1$, but we may not know where it is. The location of the object, or its pose, can be considered a hidden variable. Or in machine translation, we may know the source text $\mathbf{x}$ (say English) and the target text $\mathbf{y}$ (say French), but we typically do not know the alignment between the words.

We will extend our model as follows, to get a latent CRF:

$$p(\mathbf{y}, \mathbf{h}|\mathbf{x}, \mathbf{w}) = \frac{\exp(\mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}, \mathbf{y}, \mathbf{h}))}{Z(\mathbf{x}, \mathbf{w})} \tag{19.115}$$

$$Z(\mathbf{x}, \mathbf{w}) = \sum_{\mathbf{y}, \mathbf{h}} \exp(\mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}, \mathbf{y}, \mathbf{h})) \tag{19.116}$$

In addition, we introduce the loss function $L(\mathbf{y}^*, \mathbf{y}, \mathbf{h})$; this measures the loss when the "action" that we take is to predict $\mathbf{y}$ using latent variables $\mathbf{h}$. We could just use $L(\mathbf{y}^*, \mathbf{y})$ as before, since $\mathbf{h}$ is usually a nuisance variable and not of direct interest. However, $\mathbf{h}$ can sometimes play a useful role in defining a loss function.[12]

Given the loss function, we define our objective as

$$R_{EL}(\mathbf{w}) = -\log p(\mathbf{w}) + \sum_i \log \left[\sum_{\mathbf{y},\mathbf{h}} \exp \tilde{L}(\mathbf{y}_i, \mathbf{y}, \mathbf{h}) \frac{\exp(\mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}, \mathbf{y}, \mathbf{h}))}{Z(\mathbf{x}, \mathbf{w})} \right] \tag{19.117}$$

Using the same loose lower bound as before, we get

$$\begin{aligned} R_{EL}(\mathbf{w}) &\leq E(\mathbf{w}) + \sum_{i=1}^N \max_{\mathbf{y},\mathbf{h}} \left\{ \tilde{L}(\mathbf{y}_i, \mathbf{y}, \mathbf{h}) + \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}_i, \mathbf{y}, \mathbf{h}) \right\} \\ &\quad - \sum_{i=1}^N \max_{\mathbf{h}} \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}_i, \mathbf{y}_i, \mathbf{h}) \end{aligned} \tag{19.118}$$

If we set $E(\mathbf{w}) = -\frac{1}{2C}||\mathbf{w}||_2^2$, we get the same objective as is optimized in **latent SVMs** (Yu and Joachims 2009).

Unfortunately, this objective is no longer convex. However, it is a difference of convex functions, and hence can be solved efficiently using the **CCCP** or **concave-convex procedure** (Yuille and Rangarajan 2003). This is a method for minimizing functions of the form $f(\mathbf{w}) - g(\mathbf{w})$, where f and g are convex. The method alternates between finding a linear upper bound u on $-g$, and then minimizing the convex function $f(\mathbf{w}) + u(\mathbf{w})$; see Algorithm 6 for the pseudocode. CCCP is guaranteed to decrease the objective at every iteration, and to converge to a local minimum or a saddle point.

Algorithm 19.5: Concave-Convex Procedure (CCCP)

```
1 Set $t = 0$ and initialize $\mathbf{w}_0$ ;
2 repeat
3     Find hyperplane $\mathbf{v}_t$ such that $-g(\mathbf{w}) \leq -g(\mathbf{w}_t) + (\mathbf{w} - \mathbf{w}_t)^T \mathbf{v}_t$ for all $\mathbf{w}$ ;
4     Solve $\mathbf{w}_{t+1} = \text{argmin}_{\mathbf{w}} f(\mathbf{w}) + \mathbf{w}^T \mathbf{v}_t$ ;
5     Set $t = t + 1$
6 until converged;
```

When applied to latent SSVMs, CCCP is very similar to (hard) EM. In the "E step", we compute

12. For example, consider the problem of learning to classify a set of documents as relevant or not to a query. That is, given n documents $\mathbf{x}_1, \ldots, \mathbf{x}_n$ for a single query q, we want to produce a labeling $y_j \in \{-1, +1\}$, representing whether document j is relevant to q or not. Suppose our goal is to maximize the precision at k, which is a metric widely used in ranking (see Section 9.7.4). We will introduce a latent variable for each document h_j representing its degree of relevance. This corresponds to a latent total ordering, that has to be consistent with the observed partial ordering $\mathbf{y}$. Given this, we can define the following loss function: $L(\mathbf{y}, \hat{\mathbf{y}}, \hat{\mathbf{h}}) = \min\{1, \frac{n(\mathbf{y})}{k}\} - \frac{1}{k}\sum_{j=1}^k \mathbb{I}(y_{h_j} = 1)$, where $n(\mathbf{y})$ is the total number of relevant documents. This loss is essentially just 1 minus the precision@k, except we replace 1 with $n(\mathbf{y})/k$ so that the loss will have a minimum of zero. See (Yu and Joachims 2009) for details.

the linear upper bound by setting $\mathbf{v}_t = -C \sum_{i=1}^{N} \boldsymbol{\phi}(\mathbf{x}_i, \mathbf{y}_i, \mathbf{h}_i^*)$, where

$$\mathbf{h}_i = \underset{\mathbf{h}}{\operatorname{argmax}}\, \mathbf{w}_t^T \boldsymbol{\phi}(\mathbf{x}_i, \mathbf{y}_i, \mathbf{h}) \tag{19.119}$$

In the "M step", we estimate $\mathbf{w}$ using techniques for solving fully visible SSVMs. Specifically, we minimize

$$\frac{1}{2}||\mathbf{w}||_2 + C \sum_{i=1}^{N} \max_{\mathbf{y},\mathbf{h}} \left\{ L(\mathbf{y}_i, \mathbf{y}, \mathbf{h}) + \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}_i, \mathbf{y}, \mathbf{h}) \right\} - C \sum_{i=1}^{N} \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}_i, \mathbf{y}_i, \mathbf{h}_i^*) \tag{19.120}$$

Exercises

Exercise 19.1 Derivative of the log partition function

Derive Equation 19.40.

Exercise 19.2 CI properties of Gaussian graphical models

(Source: Jordan.)

In this question, we study the relationship between sparse matrices and sparse graphs for Gaussian graphical models. Consider a multivariate Gaussian $\mathcal{N}(x|\mu, \Sigma)$ in 3 dimensions. Suppose $\mu = (0, 0, 0)^T$ throughout.

Recall that for jointly Gaussian random variables, we know that X_i and X_j are independent iff they are uncorrelated, ie. $\Sigma_{ij} = 0$. (This is not true in general, or even if X_i and X_j are Gaussian but not jointly Gaussian.) Also, X_i is conditionally independent of X_j given all the other variables iff $\Sigma_{ij}^{-1} = 0$.

a. Suppose

$$\Sigma = \begin{pmatrix} 0.75 & 0.5 & 0.25 \\ 0.5 & 1.0 & 0.5 \\ 0.25 & 0.5 & 0.75 \end{pmatrix}$$

Are there any marginal independencies amongst X_1, X_2 and X_3? What about conditional independencies? Hint: compute Σ^{-1} and expand out $x^T \Sigma^{-1} x$: which pairwise terms $x_i x_j$ are missing? Draw an undirected graphical model that captures as many of these independence statements (marginal and conditional) as possible, but does not make any false independence assertions.

b. Suppose

$$\Sigma = \begin{pmatrix} 2 & 1 & 0 \\ 1 & 2 & 1 \\ 0 & 1 & 2 \end{pmatrix}$$

Are there any marginal independencies amongst X_1, X_2 and X_3? Are there any conditional independencies amongst X_1, X_2 and X_3? Draw an undirected graphical model that captures as many of these independence statements (marginal and conditional) as possible, but does not make any false independence assertions.

c. Now suppose the distribution on X can be represented by the following DAG:

$$X_1 \to X_2 \to X_3$$

Let the CPDs be as follows:

$$P(X_1) = \mathcal{N}(X_1; 0, 1),\ P(X_2|x_1) = \mathcal{N}(X_2; x_1, 1),\ P(X_3|x_2) = \mathcal{N}(X_3; x_2, 1) \tag{19.121}$$

Multiply these 3 CPDs together and complete the square (Bishop p101) to find the corresponding joint distribution $\mathcal{N}(X_{1:3}|\mu, \Sigma)$. (You may find it easier to solve for Σ^{-1} rather than Σ.)

d. For the DAG model in the previous question: Are there any marginal independencies amongst X_1, X_2 and X_3? What about conditional independencies? Draw an undirected graphical model that captures as many of these independence statements as possible, but does not make any false independence assertions (either marginal or conditional).

Exercise 19.3 Independencies in Gaussian graphical models

(Source: MacKay.)

a. Consider the DAG $X1 \leftarrow X2 \rightarrow X3$. Assume that all the CPDs are linear-Gaussian. Which of the following matrices *could* be the covariance matrix?

$$A = \begin{pmatrix} 9 & 3 & 1 \\ 3 & 9 & 3 \\ 1 & 3 & 9 \end{pmatrix}, B = \begin{pmatrix} 8 & -3 & 1 \\ -3 & 9 & -3 \\ 1 & -3 & 8 \end{pmatrix}, C = \begin{pmatrix} 9 & 3 & 0 \\ 3 & 9 & 3 \\ 0 & 3 & 9 \end{pmatrix}, D = \begin{pmatrix} 9 & -3 & 0 \\ -3 & 10 & -3 \\ 0 & -3 & 9 \end{pmatrix} \tag{19.122}$$

b. Which of the above matrices could be inverse covariance matrix?

c. Consider the DAG $X1 \rightarrow X2 \leftarrow X3$. Assume that all the CPDs are linear-Gaussian. Which of the above matrices could be the covariance matrix?

d. Which of the above matrices could be the inverse covariance matrix?

e. Let three variables x_1, x_2, x_4 have covariance matrix $\mathbf{\Sigma}_{(1:3)}$ and precision matrix $\mathbf{\Omega}_{(1:3)} = \mathbf{\Sigma}^{-1}_{(1:3)}$ as follows

$$\mathbf{\Sigma}_{(1:3)} = \begin{pmatrix} 1 & 0.5 & 0 \\ 0.5 & 1 & 0.5 \\ 0 & 0.5 & 1 \end{pmatrix}, \mathbf{\Omega}_{(1:3)} = \begin{pmatrix} 1.5 & -1 & 0.5 \\ -1 & 2 & -1 \\ 0.5 & -1 & 1.5 \end{pmatrix} \tag{19.123}$$

Now focus on x_1 and x_2. Which of the following statements about their covariance matrix $\mathbf{\Sigma}_{(1:2)}$ and precision matrix $\mathbf{\Omega}_{(1:2)}$ are true?

$$A : \mathbf{\Sigma}_{(1:2)} = \begin{pmatrix} 1 & 0.5 \\ 0.5 & 1 \end{pmatrix}, \ B : \mathbf{\Omega}_{(1:2)} = \begin{pmatrix} 1.5 & -1 \\ -1 & 2 \end{pmatrix} \tag{19.124}$$

Exercise 19.4 Cost of training MRFs and CRFs

(Source: Koller.) Consider the process of gradient-ascent training for a log-linear model with k features, given a data set with N training instances. Assume for simplicity that the cost of computing a single feature over a single instance in our data set is constant, as is the cost of computing the expected value of each feature once we compute a marginal over the variables in its scope. Assume that it takes c time to compute all the marginals for each data case. Also, assume that we need r iterations for the gradient process to converge.

- Using this notation, what is the time required to train an MRF in big-O notation?
- Using this notation, what is the time required to train a CRF in big-O notation?

Exercise 19.5 Full conditional in an Ising model

Consider an Ising model

$$p(x_1, \ldots, x_n|\boldsymbol{\theta}) = \frac{1}{Z(\boldsymbol{\theta})} \prod_{<ij>} \exp(J_{ij} x_i x_j) \prod_{i=1}^{n} \exp(h_i x_i) \tag{19.125}$$

where $< ij >$ denotes all unique pairs (i.e., all edges), $J_{ij} \in \mathbb{R}$ is the coupling strength (weight) on edge $i - j$, $h_i \in \mathbb{R}$ is the local evidence (bias term), and $\boldsymbol{\theta} = (\mathbf{J}, \mathbf{h})$ are all the parameters.

If $x_i \in \{0, 1\}$, derive an expression for the full conditional

$$p(x_i = 1|\mathbf{x}_{-i}, \boldsymbol{\theta}) = p(x_i = 1|\mathbf{x}_{nb_i}, \boldsymbol{\theta}) \tag{19.126}$$

where $\mathbf{x}_{-i}$ are all nodes except i, and nb_i are the neighbors of i in the graph. Hint: you answer should use the sigmoid/ logistic function $\sigma(z) = 1/(1 + e^{-z})$. Now suppose $x_i \in \{-1, +1\}$. Derive a related expression for $p(x_i|\mathbf{x}_{-i}, \boldsymbol{\theta})$ in this case. (This result can be used when applying Gibbs sampling to the model.)