Map inference

This section will explore in more detail the problem of MAP inference in graphical models. Recall that MAP inference in a graphical model p corresponds to the following optimization problem:

$$\max_x \log p(x) = \max_x \sum_c \theta_c(\mathbf{x}_c) - \log Z$$

where $\theta_c(\mathbf{x}_c) = \log \phi_c(\mathbf{x}_c)$.

In the previous section, we briefly showed how to solve this problem within the same message passing framework as marginal inference. We will now look at more efficient specialized methods.

The challenges of MAP inference

In a way, MAP inference is easier than marginal inference. One reason for this is that the intractable partition constant $\log Z$ does not depend on x and can be ignored:

$$\arg\max_x \sum_c \theta_c(\mathbf{x}_c).$$

Marginal inference can also be seen as computing and summing all assignments to the model, one of which is the MAP assignment. If we replace summation with maximization, we can also find the assignment with the highest probability; however, there exist more efficient methods than this sort of enumeration-based approach.

Note, however, that MAP inference is still not an easy problem in the general case. The above optimization objective includes many intractable problems as special cases, e.g. 3-sat. We may reduce 3-sat to MAP inference by constructing for each clause $c = (x \vee y \vee \neg z)$ a factor $\theta_c(x, y, z)$ that equals one if

x, y, z satisfy clause c, and $\theta_c(x, y, z) = 0$ otherwise. Then, the 3-sat instance is satisfiable if and only if the value of the MAP assignment equals the number of clauses[1].

Nonetheless, we will see that the MAP problem is easier than general inference, in the sense that there are some models in which MAP inference can be solved in polynomial time, while general inference is NP-hard.

Examples

Many interesting examples of MAP inference are instances of *structured prediction*, which involves doing inference in a conditional random field (CRF) model $p(y|x)$:

$$\arg\max_y \log p(y|x) = \arg\max_y \sum_c \theta_c(\mathbf{y}_c, \mathbf{x}_c).$$

$\oplus$ We discussed structured prediction in detail when we covered CRFs. Recall that our main example was handwriting recognition, in which we are given images $x_i \in [0,1]^{d \times d}$ of characters in the form of pixel matrices; MAP inference in this setting amounts to jointly recognizing the most likely word $(y_i)_{i=1}^n$ encoded by the images.

Another example of MAP inference is image segmentation; here, we are interested in locating an entity in an image and label all its pixels. Our input $\mathbf{x} \in [0,1]^{d \times d}$ is a matrix of image pixels, and our task is to predict the label $y \in \{0,1\}^{d \times d}$, indicating whether each pixel encodes the object we want to recover. Intuitively, neighboring pixels should have similar values in $\mathbf{y}$, i.e. pixels associated with the horse should form one continuous blob (rather than white noise). $\oplus$

This prior knowledge can be naturally modeled in the language of graphical models via a Potts model. As in our first example, we can introduce potentials $\phi(y_i, x)$ that encode the likelihood that any given pixel is from

our subject. We then augment them with pairwise potentials $\phi(y_i, y_j)$ for neighboring y_i, y_j, which will encourage adjacent y's to have the same value with higher probability.

Graphcuts

We will start our discussion with an efficient exact MAP inference algorithm for certain Potts models. Unlike previously-seen methods (e.g. the junction tree algorithm), this algorithm will be tractable even when the model has large treewidth.

Suppose we are given a binary pairwise MRF in which edge energies (i.e. log-edge factors) have the form

$$E_{uv}(x_u, x_v) = \begin{cases} 0 & \text{if } x_u = x_v \\ \lambda_{st} & \text{if } x_u \neq x_v, \end{cases}$$

where $\lambda_{st} \geq 0$ is a cost that penalizes edge mismatches. Assume also that nodes have unary potentials; we can always normalize the nodes' energies so that $E_u(1) = 0$ or $E_u(0) = 0$ and $E_u \geq 0$.

$\oplus$ The motivation for this model comes from image segmentation. We are looking for an assignment that minimizes the energy, which (among other things) tries to reduce discordance between adjacent variables.

We can formulate energy minimization in this type of model as a min-cut problem in an augmented graph G': we construct G' by adding special source and sink nodes s, t to our PGM graph; the node s is connected to nodes u with $E_u(0) = 0$ by an edge with weight $E_u(1)$; the node t is connected to nodes v with $E_v(1) = 0$ by an edge with weight $E_v(0)$. Finally, all the edges of the original graph get E_{uv} as their weight.

It is not hard to see that by construction, the cost of a min-cut in this graph will equal the minimum energy in the model. In particular, all nodes on the s side of the cut receive an assignment of 0, and all the ones that are on the t side receive an assignment of one. The edges between the nodes that disagree are precisely the ones that are in the min-cut.

The fastest algorithms for computing min-cuts in a graph $G = (V, E)$ take $O(EV \log V)$ or $O(V^3)$ time. Similar techniques can also be applied in slightly more general types of models with a certain type of edge potentials that are called *submodular*. We refer the reader to a textbook (e.g. Koller and Friedman) for more details.

Linear programming-based approaches

Although graphcut-based methods recover the exact MAP assignment, they are only applicable in certain restricted classes of MRFs. The algorithms we will see next solve the MAP problem approximately, but apply to much larger classes of graphical models.

Linear programming

Our first approximate inference strategy consists in reducing MAP inference to integer linear programming. Linear programming (LP) — also known as linear optimization — refers to a class of problems of the form

$$\begin{aligned} &\min \mathbf{c} \cdot \mathbf{x} \\ &\text{s.t. } A\mathbf{x} \leq \mathbf{b} \end{aligned}$$

where $\mathbf{x} \in \mathbb{R}^n$, and $\mathbf{c}, \mathbf{b} \in \mathbb{R}^{\mathbf{n}}$, $A \in \mathbb{R}^{n \times n}$ are problem parameters.

Problems of this form are found in almost every field of science and engineering. They have been extensively studied since the 30's, which has led to both an extensive theory[2], as well as practical tools that can solve very large LP instances (100,000 variables or more) in a reasonable time.

Integer linear programming (ILP) is an extension of linear programming in which we also require that $\mathbf{x} \in \{0, 1\}^n$. Unfortunately, this makes optimization considerably more difficult, and ILP is in general NP-hard. Nonetheless, there are many heuristics for solving ILP problems in practice, and commercial solvers can handle instances with thousands of variables or more.

One of the main techniques for solving ILP problems is *rounding*. The idea of rounding is to relax the requirement that $\mathbf{x} \in \{0,1\}^n$ into $0 \leq \mathbf{x} \leq 1$, solve the resulting LP, and then round the LP solution to its nearest integer value. This approach works surprisingly well in practice and has theoretical guarantees for some classes of ILPs.

Formulating MAP inference as ILP

For simplicity, let's look at MAP in pairwise MRFs. We can reduce the MAP objective to integer linear programming by introducing two types of indicator variables:

A variable $\mu_i(x_i)$ for each $i \in V$ and state x_i.

A variable $\mu_{ij}(x_i, x_j)$ for each edge $(i,j) \in E$ and pair of states x_i, x_j.

We can rewrite the MAP objective in terms of these variables as

$$\max_{\mu} \sum_{i \in V} \sum_{x_i} \theta_i(x_i)\mu_i(x_i) + \sum_{i,j \in E} \sum_{x_i, x_j} \theta_{ij}(x_i, x_j)\mu_{ij}(x_i, x_j).$$

We would like to optimize over these μ's; for that we also need to introduce constraints. First, we need to force each cluster to choose a local assignment:

$$\begin{aligned} \mu_i(x_i) &\in \{0,1\} \; \forall \, i, x_i \\ \sum_{x_i} \mu_i(x_i) &= 1 \; \forall \, i \\ \mu_{ij}(x_i, x_j) &\in \{0,1\} \; \forall \, i,j \in E, x_i, x_j \\ \sum_{x_i, x_j} \mu_{ij}(x_i, x_j) &= 1 \; \forall \, i,j \in E. \end{aligned}$$

These assignments must also be consistent:

$$\begin{aligned} \sum_{x_i} \mu_{ij}(x_i, x_j) &= \mu_j(x_j) \; \forall \, i,j \in E, x_j \\ \sum_{x_j} \mu_{ij}(x_i, x_j) &= \mu_i(x_i) \; \forall \, i,j \in E, x_i \end{aligned}$$

Together, these constraints along with the MAP objective yield an integer linear program, whose solution equals the MAP assignment. This ILP is still NP-hard, but we have an easy way to transform this into an (easy to solve) LP via relaxation. This is the essence of the linear programming approach to MAP inference.

In general, this method will only give approximate solutions. An important special case are tree-structured graphs, in which the relaxation is guaranteed to always return integer solutions, which are in turn optimal[3].

Dual decomposition

Let us now look at another way to transform the MAP objective into a more amenable optimization problem. Suppose that we are dealing with an MRF of the form

$$\max_x \sum_{i \in V} \theta_i(x_i) + \sum_{f \in F} \theta_f(x_f),$$

where F denote arbitrary factors (e.g. the edge potentials in a pairwise MRF)[4]. Let us use p^* to denote the optimal value of this objective and let x^* denote the optimal assignment.

The above objective is difficult to optimize because the potentials are coupled. Consider for a moment an alternative objective where we optimize the potentials separately:

$$\sum_{i \in V} \max_{x_i} \theta_i(x_i) + \sum_{f \in F} \max_{x^f} \theta_f(x^f).$$

This would be easy to optimize, but would only give us an upper bound on the value of the true MAP assignment. To make our relaxation tight, we would need to introduce constraints that encourage consistency between the potentials:

$$x_i^f - x_i = 0 \; \forall f, \forall i \in f.$$

The dual decomposition approach consists in softening these constraints in order to achieve a middle ground between the two optimization objective defined above.

We will achieve this by first forming the *Lagrangian* for the constrained problem, which is

$$L(\delta, \mathbf{x}^f, \mathbf{x}) = \sum_{i \in V} \theta_i(x_i) + \sum_{f \in F} \theta_f(x^f) + \sum_{f \in F} \sum_{i \in f} \sum_{x'_i} \delta_{fi}(x'_i) \left(\mathbb{I}_{x'_i = x_i} - \mathbb{I}_{x'_i = x^f_i} \right).$$

The δ variables are called Lagrange *multipliers*; each of them is associated with a constraint[5]. Observe that $x, x^f = x^*$ is a valid assignment to the Lagrangian; its value equals p^* for any δ, since the Lagrange multipliers are simply multiplied by zero. This shows that the Lagrangian is an upper bound on p^*:

$$L(\delta) := \max_{\mathbf{x}^f, \mathbf{x}} L(\delta, \mathbf{x}^f, \mathbf{x}) \geq p^* \ \forall \delta.$$

In order to get the tightest such bound, we may optimize $L(\delta)$ over δ. It turns out that by the theory of Lagarange duality, at the optimal δ^*, this bound will be exactly tight, i.e.

$$L(\delta^*) = p^*.$$

It is actually not hard to prove this in our particular setting. To see that, note that we can reparametrize the Lagrangian as:

$$\begin{aligned} L(\delta) &= \sum_{i \in V} \max_{x_i} \left(\theta_i(x_i) + \sum_{f : i \in f} \delta_{fi}(x_i) \right) + \sum_{f \in F} \max_{x^f} \left(\theta_f(x^f) + \sum_{i \in f} \delta_{fi}(x_i) \right) \\ &:= \sum_{i \in V} \max_{x_i} \bar{\theta}_i^{\delta}(x_i) + \sum_{f \in F} \max_{x^f} \bar{\theta}_f^{\delta}(x^f). \end{aligned}$$

Suppose we can find dual variables $\bar{\delta}$ such that the local maximizers of $\bar{\theta}_i^{\bar{\delta}}(x_i)$ and $\bar{\theta}_f^{\bar{\delta}}(x^f)$ agree; in other words, we can find a $\bar{x}$ such that $\bar{x}_i \in \arg\max_{x_i} \bar{\theta}_i^{\bar{\delta}}(x_i)$ and $\bar{x}^f \in \arg\max_{x^f} \bar{\theta}_f^{\bar{\delta}}(x^f)$. Then we have that

$$L(\bar{\delta}) = \sum_{i \in V} \bar{\theta}_i^{\bar{\delta}}(\bar{x}_i) + \sum_{f \in F} \bar{\theta}_f^{\bar{\delta}}(\bar{x}^f) = \sum_{i \in V} \theta_i(\bar{x}_i) + \sum_{f \in F} \theta_f(\bar{x}^f).$$

The first equality follows by definition of $L(\delta)$, while the second follows holds because terms involving Lagrange multipliers cancel out when x and x^f agree.

On the other hand, we have by the definition of p^* that

$$\sum_{i \in V} \theta_i(\bar{x}_i) + \sum_{f \in F} \theta_f(\bar{x}^f) \leq p^* \leq L(\bar{\delta})$$

which implies that $L(\bar{\delta}) = p^*$.

This argument has shown two things:

> The bound given by the Lagrangian can be made tight for the right choice of δ.
>
> To compute p^*, it suffices to find a δ at which the local sub-problems agree with each other. This happens surprisingly often in practice.

Minimizing the objective

There exist several ways of computing $L(\delta^*)$, of which we will give a brief overview.

Since the objective $L(\delta)$ is continuous and convex[6], we may minimize it using subgradient descent. Let $\bar{x}_i \in \arg\max_{x_i} \bar{\theta}_i^{\delta}(x_i)$ and let $\bar{x}^f \in \arg\max_{x^f} \bar{\theta}_f^{\delta}(x^f)$. It can be shown that the gradient $g_{fi}(x_i)$ of $L(\delta)$ w.r.t. $\delta_{fi}(x_i)$ equals 1 if $\bar{x}_i^f \neq \bar{x}_i$ and zero otherwise; similarly, $g_{fi}(x_i^f)$ equals -1 if $\bar{x}_i^f \neq \bar{x}_i$ and zero otherwise. This expression has the effect of decreasing $\bar{\theta}_i^{\delta}(\bar{x}_i)$ and increasing $\bar{\theta}_f^{\delta}(\bar{x}^f)$, thus bringing them closer to each other.

To compute these gradients we need to perform the operations $\bar{x}_i \in \arg\max_{x_i} \bar{\theta}_i^{\delta}(x_i)$ and $\bar{x}^f \in \arg\max_{x^f} \bar{\theta}_f^{\delta}(x^f)$. This is possible if the scope of the factors is small, if the graph has small tree width, if the factors are constant on most of their domain, and in many other useful special cases.

An alternative way of minimizing $L(\delta)$ is via block coordinate descent. A typical way of forming blocks is to consider all the variables $\delta_{fi}(x_i)$ associated with a fixed factor f. This results in updates that are very similar to loopy max-product belief propagation. In practice, this method may be faster than subgradient descent, is guaranteed to decrease the objective at every step, and does not require tuning a step-size parameter. Its drawback is that it does not find the global minimum (since the objective is not *strongly* convex).

Recovering the MAP assignment

As we have seen above, if a solution $\mathbf{x}, \mathbf{x}^f$ agrees on the factors for some δ, then we can guarantee that this solution is optimal.

If the optimal $\mathbf{x}, \mathbf{x}^f$ do not agree, finding the MAP assignment from this solution is still NP-hard. However, this is usually not a big problem in practice. From the point of view of theoretical guarantees, if each $\bar{\theta}_i^{\delta^*}$ has a unique maximum, then the problem will be decodable. If this guarantee is not met by all variables, we can clamp the ones that can be uniquely decoded to their optimal values and use exact inference to find the remaining variables' values.

Other methods

Local search

A more heuristic-type solution consists in starting with an arbitrary assignment and perform "moves" on the joint assignment that locally increase the probability. This technique has no guarantees; however, we can often use prior knowledge to come up with highly effective moves. Therefore, in practice, local search may perform extremely well.

Branch and bound

Alternatively, one may perform exhaustive search over the space of assignments, while pruning branches that can be provably shown not to contain a MAP assignment. The LP relaxation or its dual can be used to obtain upper bounds useful for pruning trees.

Simulated annealing

A third approach is to use sampling methods (e.g. Metropolis-Hastings) to sample from $p_t(x) \propto \exp(\frac{1}{t}\sum_{c \in C} \theta_c(x_c))$. The parameter t is called the temperature. When $t \to \infty$, p_t is close to the uniform distribution, which is easy to sample from. As $t \to 0$, p_t places more weight on $\arg\max_{\mathbf{x}} \sum_{c \in C} \theta_c(x_c)$, the quantity we want to recover. However, since this distribution is highly peaked, it is also very difficult to sample from.

The idea of simulated annealing is to run a sampling algorithm starting with a high t, and gradually decrease it, as the algorithm is being run. If the "cooling rate" is sufficiently slow, we are guaranteed to eventually find the mode of our distribution. In practice, however, choosing the rate requires a lot of tuning. This makes simulated annealing somewhat difficult to use in practice.