Inference and Representation

David Sontag

New York University

Lecture 10, Nov. 28, 2016

Reminder: conditional random fields (CRFs)

- A CRF is a Markov network on variables $\mathbf{X} \cup \mathbf{Y}$, which specifies the conditional distribution

$$P(\mathbf{y} \mid \mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_{c \in C} \phi_c(\mathbf{y}_c, \mathbf{x})$$

  with partition function

$$Z(\mathbf{x}) = \sum_{\hat{\mathbf{y}}} \prod_{c \in C} \phi_c(\hat{\mathbf{y}}_c, \mathbf{x}).$$

- As before, two variables in the graph are connected with an undirected edge if they appear together in the scope of some factor
- The only difference with a standard Markov network is the normalization term – before marginalized over $\mathbf{X}$ and $\mathbf{Y}$, now only over $\mathbf{Y}$

Reminder: log-linear parameterization of CRFs

- Parameterize $\phi_c(\mathbf{y}_c, \mathbf{x})$ using a log-linear parameterization:
  - Single weight vector $\mathbf{w} \in \mathbb{R}^d$ that is used globally
  - For each potential c, a vector-valued **feature function** $\mathbf{f}_c(\mathbf{y}_c, \mathbf{x}) \in \mathbb{R}^d$
  - Then, $\phi_c(\mathbf{y}_c, \mathbf{x}; \mathbf{w}) = \exp(\mathbf{w} \cdot \mathbf{f}_c(\mathbf{y}_c, \mathbf{x}))$
- The conditional distribution is in the *exponential family*!

$$p(\mathbf{y} \mid \mathbf{x}; \mathbf{w}) = \exp\{\mathbf{w} \cdot \mathbf{f}(\mathbf{y}, \mathbf{x}) - \log Z(\mathbf{x}; \mathbf{w})\},$$

where $\mathbf{f}(\mathbf{y}, \mathbf{x}) = \sum_c \mathbf{f}_c(\mathbf{y}_c, \mathbf{x})$ and $Z(\mathbf{x}; \mathbf{w}) = \sum_{\hat{\mathbf{y}}} \exp\{\mathbf{w} \cdot \mathbf{f}(\hat{\mathbf{y}}, \mathbf{x})\}$

- This formulation allows for parameter sharing

Reminder: density estimation for CRFs

$$\text{CRF:} \quad p(\mathbf{y} \mid \mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_{c \in C} \phi_c(\mathbf{y}_c, \mathbf{x}), \quad Z(\mathbf{x}) = \sum_{\hat{\mathbf{y}}} \prod_{c \in C} \phi_c(\hat{\mathbf{y}}_c, \mathbf{x})$$

- Empirical risk minimization with CRFs, i.e. $\min_{\hat{\mathcal{M}}} \mathbf{E}_{\mathcal{D}} \left[loss(\mathbf{x}, \mathbf{y}, \hat{\mathcal{M}}) \right]$:

$$\begin{aligned}
\mathbf{w}^{ML} &= \arg\min_{\mathbf{w}} \frac{1}{|\mathcal{D}|} \sum_{(\mathbf{x},\mathbf{y}) \in \mathcal{D}} -\log p(\mathbf{y} \mid \mathbf{x}; \mathbf{w}) \\
&= \arg\max_{\mathbf{w}} \sum_{(\mathbf{x},\mathbf{y}) \in \mathcal{D}} \left(\sum_c \log \phi_c(\mathbf{y}_c, \mathbf{x}; \mathbf{w}) - \log Z(\mathbf{x}; \mathbf{w}) \right) \\
&= \arg\max_{\mathbf{w}} \mathbf{w} \cdot \left(\sum_{(\mathbf{x},\mathbf{y}) \in \mathcal{D}} \sum_c \mathbf{f}_c(\mathbf{y}_c, \mathbf{x}) \right) - \sum_{(\mathbf{x},\mathbf{y}) \in \mathcal{D}} \log Z(\mathbf{x}; \mathbf{w})
\end{aligned}$$

- What if prediction is only done with MAP inference? Then, the partition function is irrelevant. Is there a way to train to take advantage of this?

Goal of learning

- The goal of learning is to return a model $\hat{\mathcal{M}}$ that precisely captures the distribution p^* from which our data was sampled
- This is in general not achievable because of
  - computational reasons
  - limited data only provides a rough approximation of the true underlying distribution
- We need to select $\hat{\mathcal{M}}$ to construct the "best" approximation to $\mathcal{M}^*$
- What is "best"?

What notion of “best” should learning be optimizing?

This depends on what we want to do

1. Density estimation: we are interested in the full distribution (so later we can compute whatever conditional probabilities we want)
2. Specific prediction tasks: we are using the distribution to make a prediction
3. Structure or knowledge discovery: we are interested in the model itself

Structured prediction

- Often we learn a model for the purpose of **structured prediction**, in which given $\mathbf{x}$ we predict $\mathbf{y}$ by finding the MAP assignment:

$$\operatorname*{argmax}_{\mathbf{y}} \hat{p}(\mathbf{y}|\mathbf{x})$$

- Rather than learn using log-loss (density estimation), we use a loss function better suited to the specific task
- One reasonable choice would be the **classification error**:

$$\mathbf{E}_{(\mathbf{x},\mathbf{y})\sim p^*}\left[\mathbb{1}\{\ \exists \mathbf{y}' \neq \mathbf{y} \text{ s.t. } \hat{p}(\mathbf{y}'|\mathbf{x}) \geq \hat{p}(\mathbf{y}|\mathbf{x})\ \}\right]$$

  which is the probability over all $(\mathbf{x}, \mathbf{y})$ pairs sampled from p^* that our classifier selects the right labels

- If p^* is in the model family, training with log-loss (density estimation) and classification error would perform similarly (given sufficient data)
- Otherwise, better to directly go for what we care about (classification error)

Structured prediction

- Consider the empirical risk for 0-1 loss (classification error):

$$\frac{1}{|\mathcal{D}|} \sum_{(\mathbf{x},\mathbf{y})\in\mathcal{D}} \mathbb{1}\{\ \exists \mathbf{y}' \neq \mathbf{y} \text{ s.t. } \hat{p}(\mathbf{y}'|\mathbf{x}) \geq \hat{p}(\mathbf{y}|\mathbf{x})\ \}$$

- Each constraint $\hat{p}(\mathbf{y}'|\mathbf{x}) \geq \hat{p}(\mathbf{y}|\mathbf{x})$ is equivalent to

$$\mathbf{w} \cdot \sum_c \mathbf{f}_c(\mathbf{x}, \mathbf{y}'_c) - \log Z(\mathbf{x}; \mathbf{w}) \geq \mathbf{w} \cdot \sum_c \mathbf{f}_c(\mathbf{x}, \mathbf{y}_c) - \log Z(\mathbf{x}; \mathbf{w})$$

- The log-partition function cancels out on both sides. Re-arranging, we have:

$$\mathbf{w} \cdot \left(\sum_c \mathbf{f}_c(\mathbf{x}, \mathbf{y}'_c) - \sum_c \mathbf{f}_c(\mathbf{x}, \mathbf{y}_c) \right) \geq 0$$

- Said differently, the empirical risk is **zero** when $\forall (\mathbf{x}, \mathbf{y}) \in \mathcal{D}$ and $\mathbf{y}' \neq \mathbf{y}$,

$$\mathbf{w} \cdot \left(\sum_c \mathbf{f}_c(\mathbf{x}, \mathbf{y}_c) - \sum_c \mathbf{f}_c(\mathbf{x}, \mathbf{y}'_c) \right) > 0.$$

Structured prediction

- Empirical risk is **zero** when $\forall(\mathbf{x},\mathbf{y}) \in \mathcal{D}$ and $\mathbf{y}' \neq \mathbf{y}$,

$$\mathbf{w} \cdot \left(\sum_c \mathbf{f}_c(\mathbf{x},\mathbf{y}_c) - \sum_c \mathbf{f}_c(\mathbf{x},\mathbf{y}'_c) \right) > 0.$$

- In the simplest setting, learning corresponds to finding a weight vector **w** that satisfies all of these constraints (when possible)
- This is a linear program (LP)!
- How many constraints does it have? $|\mathcal{D}| * |\mathcal{Y}|$ – exponentially many!
- Thus, we must avoid explicitly representing this LP
- The first part of this lecture is about algorithms for solving this LP (or some variant) in a tractable manner

Structured *perceptron* algorithm

- **Input:** Training examples $\mathcal{D} = \{(\mathbf{x}^m, \mathbf{y}^m)\}$
- Let $\mathbf{f}(\mathbf{x}, \mathbf{y}) = \sum_c \mathbf{f}_c(\mathbf{x}, \mathbf{y}_c)$. Then, the constraints that we want to satisfy are

$$\mathbf{w} \cdot \Big(\mathbf{f}(\mathbf{x}^m, \mathbf{y}^m) - \mathbf{f}(\mathbf{x}^m, \mathbf{y})\Big) > 0, \quad \forall \mathbf{y} \neq \mathbf{y}^m$$

- The perceptron algorithm uses MAP inference in its inner loop:

$$\mathrm{MAP}(\mathbf{x}^m; \mathbf{w}) = \arg\max_{\mathbf{y} \in \mathcal{Y}} \mathbf{w} \cdot \mathbf{f}(\mathbf{x}^m, \mathbf{y})$$

  The maximization can often be performed efficiently by using the structure!

- The perceptron algorithm is then:

```
1 Start with $\mathbf{w} = 0$
2 While the weight vector is still changing:
3     For $m = 1, \ldots, |\mathcal{D}|$
4         $\mathbf{y} \leftarrow \mathrm{MAP}(\mathbf{x}^m; \mathbf{w})$
5         $\mathbf{w} \leftarrow \mathbf{w} + \mathbf{f}(\mathbf{x}^m, \mathbf{y}^m) - \mathbf{f}(\mathbf{x}^m, \mathbf{y})$
```

Structured perceptron algorithm

- If the training data is *separable*, the perceptron algorithm is guaranteed to find a weight vector which perfectly classifies all of the data
- When separable with margin γ, number of iterations is at most

$$\left(\frac{2R}{\gamma}\right)^2,$$

  where $R = \max_{m,\mathbf{y}} ||\mathbf{f}(\mathbf{x}^m, \mathbf{y})||_2$
- In practice, one stops after a certain number of outer iterations (called *epochs*), and uses the *average* of all weights
- The averaging can be understood as a type of regularization to prevent overfitting

Allowing slack

- We can equivalently write the constraints as

$$\mathbf{w} \cdot \Big(\mathbf{f}(\mathbf{x}^m, \mathbf{y}^m) - \mathbf{f}(\mathbf{x}^m, \mathbf{y})\Big) \geq 1, \quad \forall \mathbf{y} \neq \mathbf{y}^m$$

- Suppose there do not exist weights **w** that satisfy all constraints
- Introduce *slack* variables $\xi_m \geq 0$, one per data point, to allow for constraint violations:

$$\mathbf{w} \cdot \Big(\mathbf{f}(\mathbf{x}^m, \mathbf{y}^m) - \mathbf{f}(\mathbf{x}^m, \mathbf{y})\Big) \geq 1 - \xi_m, \quad \forall \mathbf{y} \neq \mathbf{y}^m$$

- Then, minimize the sum of the slack variables, $\min_{\xi \geq 0} \sum_m \xi_m$, subject to the above constraints

Structural SVM (support vector machine)

$$\min_{\mathbf{w},\xi} \quad \sum_m \xi_m + C||\mathbf{w}||^2$$

subject to:

$$\begin{aligned} \mathbf{w} \cdot \Big(\mathbf{f}(\mathbf{x}^m, \mathbf{y}^m) - \mathbf{f}(\mathbf{x}^m, \mathbf{y})\Big) &\geq 1 - \xi_m, \quad \forall m, \mathbf{y} \neq \mathbf{y}^m \\ \xi_m &\geq 0, \quad \forall m \end{aligned}$$

This is a quadratic program (QP). Solving for the slack variables in closed form, we obtain

$$\xi_m^* = \max\left(0, \max_{\mathbf{y} \in \mathcal{Y} \setminus \mathbf{y}^m} 1 - \mathbf{w} \cdot \Big(\mathbf{f}(\mathbf{x}^m, \mathbf{y}^m) - \mathbf{f}(\mathbf{x}^m, \mathbf{y})\Big)\right)$$

Thus, we can re-write the whole optimization problem as

$$\min_{\mathbf{w}} \sum_m \max\left(0, \max_{\mathbf{y} \in \mathcal{Y} \setminus \mathbf{y}^m} 1 - \mathbf{w} \cdot \Big(\mathbf{f}(\mathbf{x}^m, \mathbf{y}^m) - \mathbf{f}(\mathbf{x}^m, \mathbf{y})\Big)\right) + C||\mathbf{w}||^2$$

Hinge loss

- We can view $\max\left(0, \ \max_{\mathbf{y} \in \mathcal{Y} \setminus \mathbf{y}^m} \ 1 - \mathbf{w} \cdot \left(\mathbf{f}(\mathbf{x}^m, \mathbf{y}^m) - \mathbf{f}(\mathbf{x}^m, \mathbf{y})\right)\right)$ as a loss function, called *hinge loss*
- When $\mathbf{w} \cdot \mathbf{f}(\mathbf{x}^m, \mathbf{y}^m) \geq \mathbf{w} \cdot \mathbf{f}(\mathbf{x}^m, \mathbf{y})$ for all $\mathbf{y}$ (i.e., correct prediction), this takes a value between 0 and 1
- When $\exists \mathbf{y} \neq \mathbf{y}^m$ such that $\mathbf{w} \cdot \mathbf{f}(\mathbf{x}^m, \mathbf{y}) \geq \mathbf{w} \cdot \mathbf{f}(\mathbf{x}^m, \mathbf{y}^m)$ (i.e., incorrect prediction), this takes a value ≥ 1
- Thus, this always *upper bounds* the 0-1 loss!
- Minimizing hinge loss is good because it minimizes an upper bound on the 0-1 loss (prediction error)

Better Metrics

- It doesn't always make sense to penalize all incorrect predictions equally!
- We can change the constraints to

$$\mathbf{w} \cdot \Big(\mathbf{f}(\mathbf{x}^m, \mathbf{y}^m) - \mathbf{f}(\mathbf{x}^m, \mathbf{y})\Big) \geq \Delta(\mathbf{y}, \mathbf{y}^m) - \xi_m, \quad \forall \mathbf{y},$$

  where $\Delta(\mathbf{y}, \mathbf{y}^m) \geq 0$ is a measure of how far the assignment $\mathbf{y}$ is from the true assignment $\mathbf{y}^m$
- This is called **margin scaling** (as opposed to *slack scaling*)
- We assume that $\Delta(\mathbf{y}, \mathbf{y}) = 0$, which allows us to say that the constraint holds for *all* $\mathbf{y}$, rather than just $\mathbf{y} \neq \mathbf{y}^m$
- A frequently used metric for MRFs is **Hamming distance**, where $\Delta(\mathbf{y}, \mathbf{y}^m) = \sum_{i \in V} \mathbb{1}[y_i \neq y_i^m]$

Structural SVM with margin scaling

$$\min_{\mathbf{w}} \sum_{m} \max_{\mathbf{y}\in\mathcal{Y}} \left(\Delta(\mathbf{y},\mathbf{y}^m) - \mathbf{w}\cdot\left(\mathbf{f}(\mathbf{x}^m,\mathbf{y}^m) - \mathbf{f}(\mathbf{x}^m,\mathbf{y})\right)\right) + C||\mathbf{w}||^2$$

How to solve this? Many methods!

1. Cutting-plane algorithm (Tsochantaridis et al., 2005)
2. Stochastic subgradient method (Ratliff et al., 2007)
3. Dual Loss Primal Weights algorithm (Meshi et al., 2010)
4. Frank-Wolfe algorithm (Lacoste-Julien et al., 2013)

Stochastic subgradient method

$$\min_{\mathbf{w}} \sum_{m} \max_{\mathbf{y}\in\mathcal{Y}} \Big(\Delta(\mathbf{y},\mathbf{y}^m) - \mathbf{w}\cdot\Big(\mathbf{f}(\mathbf{x}^m,\mathbf{y}^m) - \mathbf{f}(\mathbf{x}^m,\mathbf{y})\Big)\Big) + C||\mathbf{w}||^2$$

- Although this objective is convex, it is not differentiable everywhere
- We can use a *subgradient* method to minimize (instead of gradient descent)
- The *subgradient* of $\max_{\mathbf{y}\in\mathcal{Y}} \Delta(\mathbf{y},\mathbf{y}^m) - \mathbf{w}\cdot\Big(\mathbf{f}(\mathbf{x}^m,\mathbf{y}^m) - \mathbf{f}(\mathbf{x}^m,\mathbf{y})\Big)$ at $\mathbf{w}^{(t)}$ is

  $$\mathbf{f}(\mathbf{x}^m,\hat{\mathbf{y}}) - \mathbf{f}(\mathbf{x}^m,\mathbf{y}^m),$$

  where $\hat{\mathbf{y}}$ is one of the maximizers with respect to $\mathbf{w}^{(t)}$, i.e.

  $$\hat{\mathbf{y}} = \arg\max_{\mathbf{y}\in\mathcal{Y}} \Delta(\mathbf{y},\mathbf{y}^m) + \mathbf{w}^{(t)}\cdot\mathbf{f}(\mathbf{x}^m,\mathbf{y})$$

- This maximization is called *loss-augmented* MAP inference

Loss-augmented inference

$$\hat{\mathbf{y}} = \arg\max_{\mathbf{y}\in\mathcal{Y}} \ \Delta(\mathbf{y}, \mathbf{y}^m) + \mathbf{w}^{(t)} \cdot \mathbf{f}(\mathbf{x}^m, \mathbf{y})$$

- When $\Delta(\mathbf{y}, \mathbf{y}^m) = \sum_{i\in V} \mathbb{1}[y_i \neq y_i^m]$, this corresponds to adding additional single-node potentials

$$\theta_i(y_i) = 1 \text{ if } y_i \neq y_i^m, \text{ and } 0 \text{ otherwise}$$

- If MAP inference was previously exactly solvable by a combinatorial algorithm, loss-augmented MAP inference typically is too
- The Hamming distance pushes the MAP solution *away* from the true assignment $\mathbf{y}^m$

Cutting-plane algorithm

$$\min_{\mathbf{w},\xi} \sum_m \xi_m + C||\mathbf{w}||^2$$

subject to:

$$\begin{aligned} \mathbf{w} \cdot \Big(\mathbf{f}(\mathbf{x}^m, \mathbf{y}^m) - \mathbf{f}(\mathbf{x}^m, \mathbf{y})\Big) &\geq \Delta(\mathbf{y}, \mathbf{y}^m) - \xi_m, \quad \forall m, \mathbf{y} \in \mathcal{Y}_m \\ \xi_m &\geq 0, \quad \forall m \end{aligned}$$

- Start with $\mathcal{Y}_m = \{\mathbf{y}^m\}$. Solve for the optimal $\mathbf{w}^*, \xi^*$
- Then, look to see if any of the *unused* constraints are violated
- To find a violated constraint for data point m, simply solve the loss-augmented inference problem:

$$\hat{\mathbf{y}} = \arg\max_{\mathbf{y} \in \mathcal{Y}} \Delta(\mathbf{y}, \mathbf{y}^m) + \mathbf{w} \cdot \mathbf{f}(\mathbf{x}^m, \mathbf{y})$$

- If $\hat{\mathbf{y}} \in \mathcal{Y}_m$, do nothing. Otherwise, let $\mathcal{Y}_m = \mathcal{Y}_m \cup \{\hat{\mathbf{y}}\}$
- Repeat until no new constraints are added. Then we are optimal!

Cutting-plane algorithm

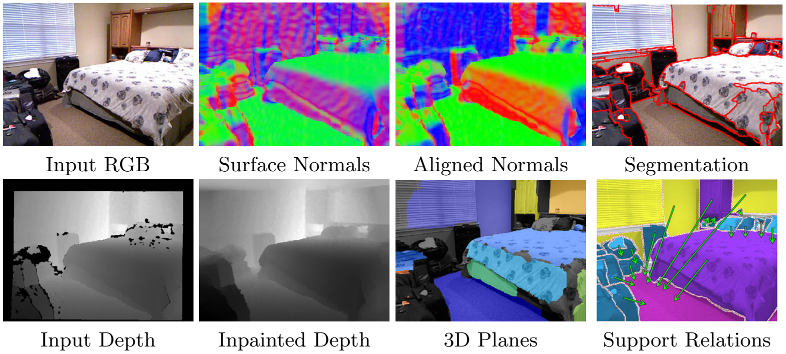
- Can prove that, in order to solve the structural SVM up to ϵ (additive) accuracy, takes a polynomial number of iterations
- In practice, terminates very quickly

Summary of convergence rates

Optimization algorithm	Online	Primal/Dual	Type of guarantee	Oracle type	# Oracle calls
dual extragradient (Taskar et al., 2006)	no	primal-'dual'	saddle point gap	Bregman projection	$O\left(\frac{nR\log\lvert\mathcal{Y}\rvert}{\lambda\varepsilon}\right)$
online exponentiated gradient (Collins et al., 2008)	yes	dual	expected dual error	expectation	$O\left(\frac{(n+\log\lvert\mathcal{Y}\rvert)R^2}{\lambda\varepsilon}\right)$
excessive gap reduction (Zhang et al., 2011)	no	primal-dual	duality gap	expectation	$O\left(nR\sqrt{\frac{\log\lvert\mathcal{Y}\rvert}{\lambda\varepsilon}}\right)$
BMRM (Teo et al., 2010)	no	primal	$\geq$primal error	maximization	$O\left(\frac{nR^2}{\lambda\varepsilon}\right)$
1-slack SVM-Struct (Joachims et al., 2009)	no	primal-dual	duality gap	maximization	$O\left(\frac{nR^2}{\lambda\varepsilon}\right)$
stochastic subgradient (Shalev-Shwartz et al., 2010a)	yes	primal	primal error w.h.p.	maximization	$\tilde{O}\left(\frac{R^2}{\lambda\varepsilon}\right)$
this paper: block-coordinate Frank-Wolfe	yes	primal-dual	expected duality gap	maximization	$O\left(\frac{R^2}{\lambda\varepsilon}\right)$ Thm. 3

R same as before. n=number of training examples. λ is the regularization constant (correpsonding to $2C/n$)

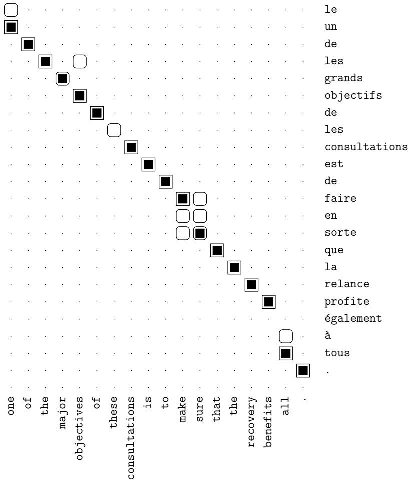
Application to segmentation & support inference



(Silberman, Sontag, Fergus. ECCV '14)

Application to machine translation

Word alignment between languages:



(Taskar, Lacoste-Julien, Klein. EMNLP '05)

MAP inference

- Recall the MAP inference task,

$$\arg\max_{\mathbf{x}} p(\mathbf{x}), \qquad p(\mathbf{x}) = \frac{1}{Z}\prod_{c\in C}\phi_c(\mathbf{x}_c)$$

  (we assume any evidence has been subsumed into the potentials)

- Since the normalization term is simply a constant, this is equivalent to

$$\arg\max_{\mathbf{x}} \prod_{c\in C}\phi_c(\mathbf{x}_c)$$

  (called the *max-product* inference task)

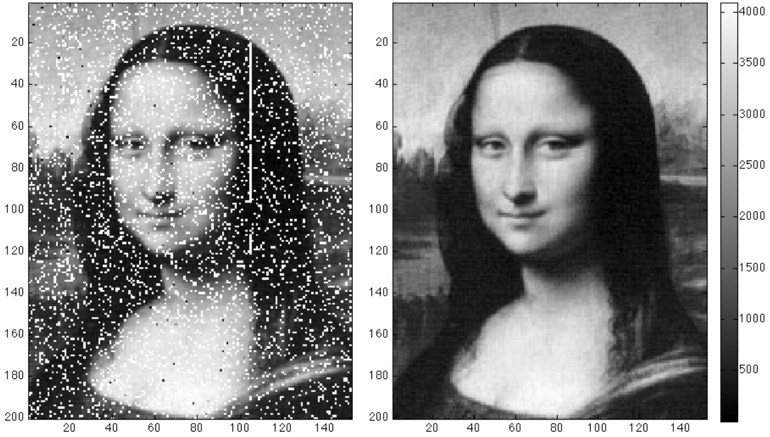
- Furthermore, since log is monotonic, letting $\theta_c(\mathbf{x_c}) = \lg \phi_c(\mathbf{x_c})$, we have that this is equivalent to

$$\arg\max_{\mathbf{x}} \sum_{c\in C}\theta_c(\mathbf{x}_c)$$

  (called *max-sum*)

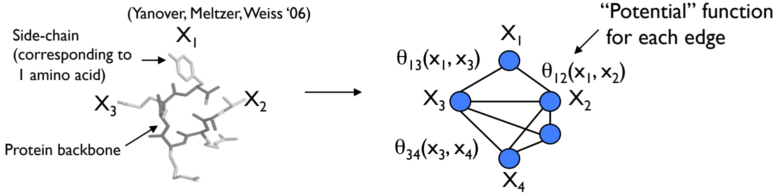
Motivating application: image denoising

- Input (left): noisy image
- Output (right): denoised image



Motivating application: protein side-chain placement

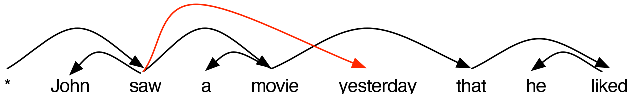
- Find "minimum energy" conformation of amino acid side-chains along a fixed carbon backbone:



- Orientations of the side-chains are represented by discretized angles called rotamers
- Rotamer choices for nearby amino acids are energetically coupled (attractive and repulsive forces)

Motivating application: dependency parsing

- Given a sentence, predict the dependency tree that relates the words:



- Arc from head word of each phrase to words that modify it
- May be *non-projective*: each word and its descendents may not be a contiguous subsequence
- m words $\Longrightarrow m(m-1)$ binary arc selection variables $x_{ij} \in \{0,1\}$
- Let $\mathbf{x}_{|i} = \{x_{ij}\}_{j \neq i}$ (all outgoing edges). Predict with:

$$\max_{\mathbf{x}} \theta_T(\mathbf{x}) + \sum_{ij} \theta_{ij}(x_{ij}) + \sum_{i} \theta_{i|}(\mathbf{x}_{|i})$$

How to perform approximate MAP inference?

- Local search (iterated conditional modes)
  - Start from an arbitrary assignment (e.g., random). Iterate:
  - Choose a variable. Change a new state for this variable to maximize the value of the resulting assignment
- Branch-and-bound
  - Exhaustive search over space of assignments, pruning branches that can be provably shown not to contain a MAP assignment
  - Can use LP relaxations or dual decomposition to obtain upper bounds
  - Lower bound obtained from value of any assignment found
- Branch-and-cut (most powerful method; used by CPLEX & Gurobi)
  - Same as branch-and-bound; spend more time getting tighter bounds
  - Adds *new constraints* to cut off fractional solutions of the LP relaxation, making the upper bound tighter

Dual decomposition

effective, simple, approximate MAP inference, with guarantees

- Consider the MAP problem for pairwise Markov random fields:

$$\mathrm{MAP}(\theta) = \max_{\mathbf{x}} \sum_{i \in V} \theta_i(x_i) + \sum_{ij \in E} \theta_{ij}(x_i, x_j).$$

- If we push the maximizations *inside* the sums, the value can only *increase*:

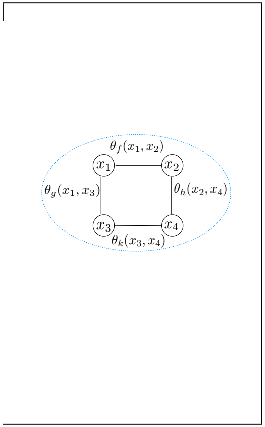
$$\mathrm{MAP}(\theta) \leq \sum_{i \in V} \max_{x_i} \theta_i(x_i) + \sum_{ij \in E} \max_{x_i, x_j} \theta_{ij}(x_i, x_j)$$

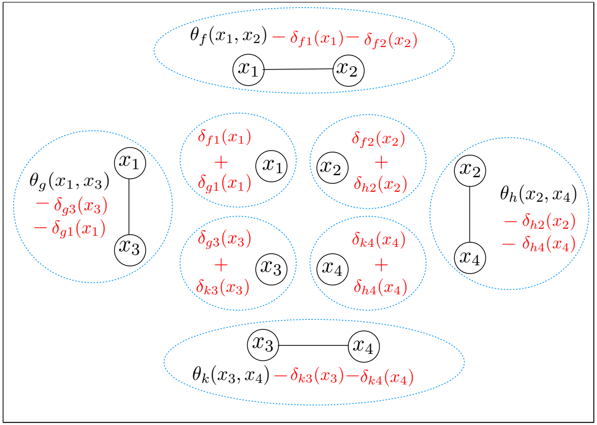
- Note that the right-hand side can be easily evaluated
- One can always *reparameterize* a distribution by operations like

$$\begin{aligned} \theta_i^{\mathrm{new}}(x_i) &= \theta_i^{\mathrm{old}}(x_i) + f(x_i) \\ \theta_{ij}^{\mathrm{new}}(x_i, x_j) &= \theta_{ij}^{\mathrm{old}}(x_i, x_j) - f(x_i) \end{aligned}$$

for **any** function $f(x_i)$, without changing the distribution/energy

Dual decomposition





$\theta_f(x_1,x_2) - \delta_{f1}(x_1) - \delta_{f2}(x_2)$
x_1
x_2
$\theta_g(x_1,x_3) - \delta_{g3}(x_3) - \delta_{g1}(x_1)$
x_1
x_3
$\delta_{f1}(x_1) + \delta_{g1}(x_1)$
x_1
$\delta_{f2}(x_2) + \delta_{h2}(x_2)$
x_2
$\theta_h(x_2,x_4) - \delta_{h2}(x_2) - \delta_{h4}(x_4)$
x_2
x_4
$\delta_{g3}(x_3) + \delta_{k3}(x_3)$
x_3
$\delta_{k4}(x_4) + \delta_{h4}(x_4)$
x_4
x_3
x_4
$\theta_k(x_3,x_4) - \delta_{k3}(x_3) - \delta_{k4}(x_4)$

Dual decomposition

- Define:

$$\begin{aligned}\tilde{\theta}_i(x_i) &= \theta_i(x_i) + \sum_{ij \in E} \delta_{j \to i}(x_i) \\ \tilde{\theta}_{ij}(x_i, x_j) &= \theta_{ij}(x_i, x_j) - \delta_{j \to i}(x_i) - \delta_{i \to j}(x_j)\end{aligned}$$

- It is easy to verify that

$$\sum_i \theta_i(x_i) + \sum_{ij \in E} \theta_{ij}(x_i, x_j) = \sum_i \tilde{\theta}_i(x_i) + \sum_{ij \in E} \tilde{\theta}_{ij}(x_i, x_j) \quad \forall \mathbf{x}$$

- Thus, we have that:

$$\mathrm{MAP}(\theta) = \mathrm{MAP}(\tilde{\theta}) \leq \sum_{i \in V} \max_{x_i} \tilde{\theta}_i(x_i) + \sum_{ij \in E} \max_{x_i, x_j} \tilde{\theta}_{ij}(x_i, x_j)$$

- Every value of δ gives a different upper bound on the value of the MAP!
- The **tightest** upper bound can be obtained by minimizing the r.h.s. with respect to δ!

Dual decomposition

- We obtain the following **dual** objective: $L(\delta) =$

$$\sum_{i \in V} \max_{x_i} \Big(\theta_i(x_i) + \sum_{ij \in E} \delta_{j \to i}(x_i) \Big) + \sum_{ij \in E} \max_{x_i, x_j} \Big(\theta_{ij}(x_i, x_j) - \delta_{j \to i}(x_i) - \delta_{i \to j}(x_j) \Big),$$

$$\text{DUAL-LP}(\theta) = \min_{\delta} L(\delta)$$

- This provides an upper bound on the MAP assignment!

$$\text{MAP}(\theta) \quad \leq \quad \text{DUAL-LP}(\theta) \leq L(\delta)$$

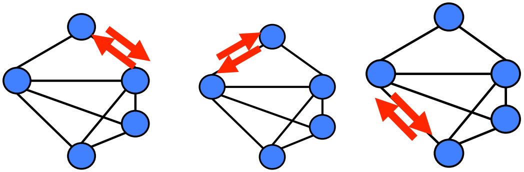
- How can find δ which give tight bounds?

Solving the dual efficiently

- Many ways to solve the dual linear program, i.e. minimize with respect to δ:

$$\sum_{i \in V} \max_{x_i} \Big(\theta_i(x_i) + \sum_{ij \in E} \delta_{j \to i}(x_i) \Big) + \sum_{ij \in E} \max_{x_i, x_j} \Big(\theta_{ij}(x_i, x_j) - \delta_{j \to i}(x_i) - \delta_{i \to j}(x_j) \Big),$$

- One option is to use the subgradient method
- Can also solve using **block coordinate-descent**, which gives algorithms that look very much like belief propagation:



Max-product linear programming (MPLP) algorithm

Input: A set of factors $\theta_i(x_i), \theta_{ij}(x_i, x_j)$

Output: An assignment $x_1, \ldots, x_n$ that approximates the MAP

Algorithm:

- Initialize $\delta_{i\to j}(x_j) = 0, \quad \delta_{j\to i}(x_i) = 0, \quad \forall ij \in E, x_i, x_j$
- Iterate until small enough change in $L(\delta)$:

  For each edge $ij \in E$ (sequentially), perform the updates:

$$\begin{aligned}
\delta_{j\to i}(x_i) &= -\frac{1}{2}\delta_i^{-j}(x_i) + \frac{1}{2}\max_{x_j}\left[\theta_{ij}(x_i, x_j) + \delta_j^{-i}(x_j)\right] \quad \forall x_i \\
\delta_{i\to j}(x_j) &= -\frac{1}{2}\delta_j^{-i}(x_j) + \frac{1}{2}\max_{x_i}\left[\theta_{ij}(x_i, x_j) + \delta_i^{-j}(x_i)\right] \quad \forall x_j
\end{aligned}$$

  where $\delta_i^{-j}(x_i) = \theta_i(x_i) + \sum_{ik\in E, k\neq j} \delta_{k\to i}(x_i)$

- Return $x_i \in \arg\max_{\hat{x}_i} \tilde{\theta}_i^{\delta}(\hat{x}_i)$

Generalization to arbitrary factor graphs

Inputs:

- A set of factors $\theta_i(x_i), \theta_f(\boldsymbol{x}_f)$.

Output:

- An assignment $x_1, \ldots, x_n$ that approximates the MAP.

Algorithm:

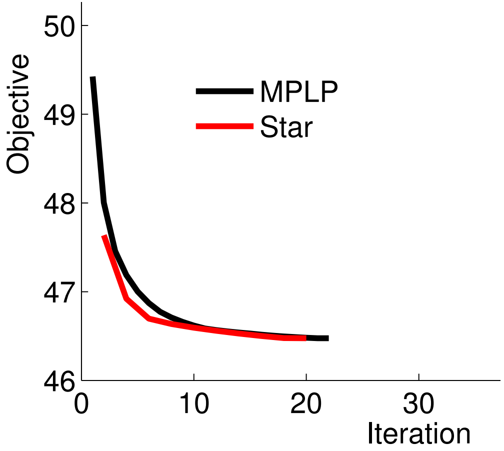
- Initialize $\delta_{fi}(x_i) = 0, \quad \forall f \in F, i \in f, x_i$.
- Iterate until small enough change in $L(\boldsymbol{\delta})$ (see Eq. 1.2):
  For each $f \in F$, perform the updates

$$\delta_{fi}(x_i) = -\delta_i^{-f}(x_i) + \frac{1}{|f|} \max_{\boldsymbol{x}_{f \setminus i}} \left[\theta_f(\boldsymbol{x}_f) + \sum_{\hat{i} \in f} \delta_{\hat{i}}^{-f}(x_{\hat{i}}) \right], \tag{1.16}$$

  simultaneously for all $i \in f$ and x_i. We define $\delta_i^{-f}(x_i) = \theta_i(x_i) + \sum_{\hat{f} \neq f} \delta_{\hat{f}i}(x_i)$.
- Return $x_i \in \arg\max_{\hat{x}_i} \bar{\theta}_i^{\boldsymbol{\delta}}(\hat{x}_i)$ (see Eq. 1.6).

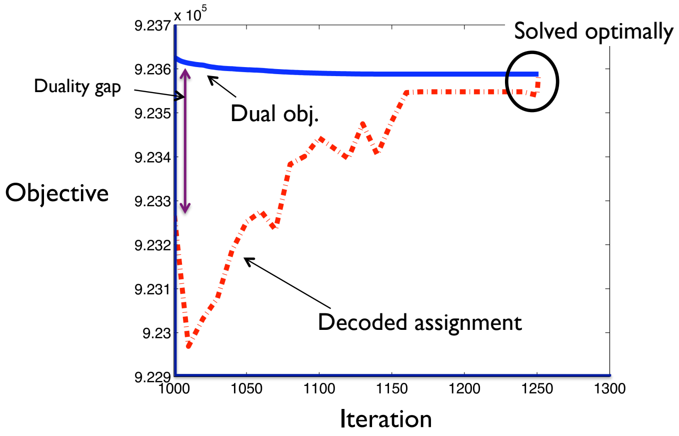
Experimental results

Comparison of two block coordinate descent algorithms on a 10×10 node Ising grid:



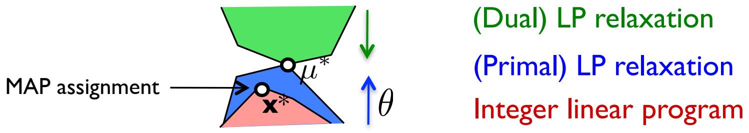
Experimental results

Performance on stereo vision inference task:



Linear programming duality

Beyond the scope of this class, but one can show intimate relationship between dual decomposition and linear programming relaxations:



$$\mathrm{MAP}(\theta) \le \mathrm{LP}(\theta) = \mathrm{DUAL\text{-}LP}(\theta) \le L(\delta)$$