Variational Inference for Structured NLP Models: Quick Reference

Berkeley NLP

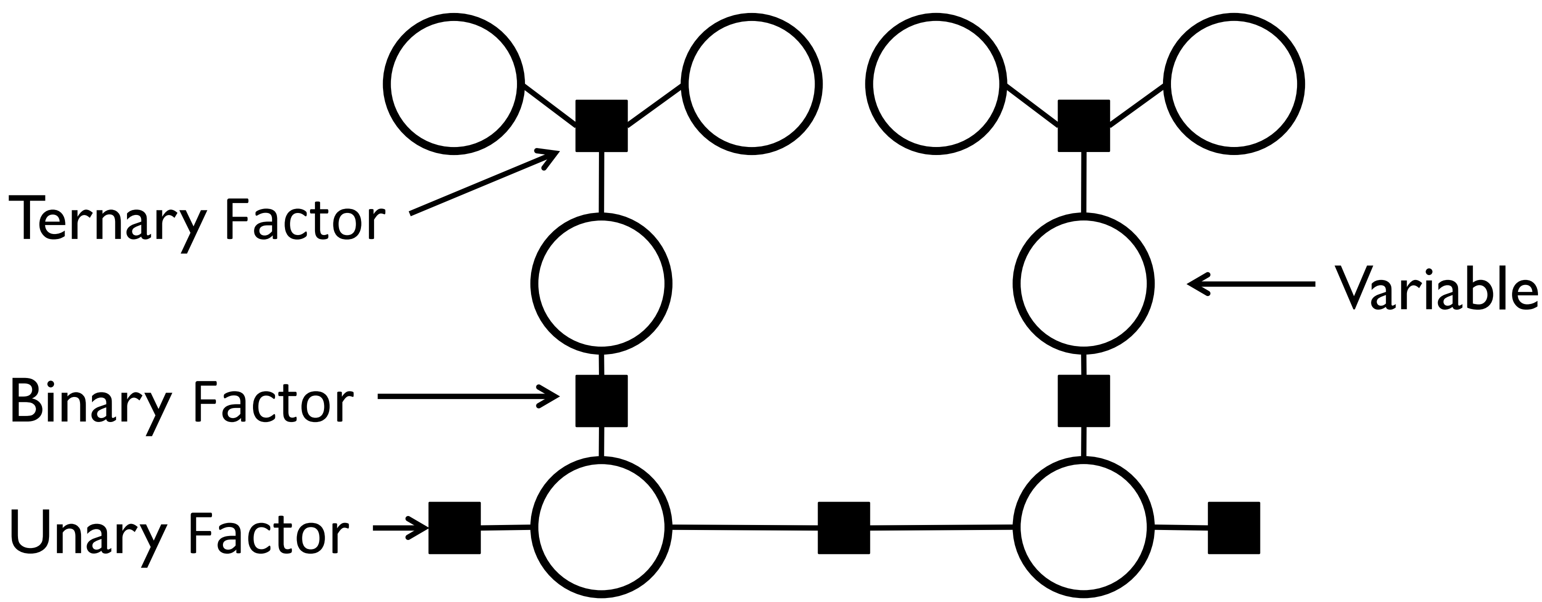
NAACL-HLT, June 3, 2012

David Burkett and Dan Klein

(full slides available at http://nlp.cs.berkeley.edu/tutorials)

Models

- All models we discuss can be represented as discrete Markov Random Fields
- We use factor graph notation, explicitly separating variables and factors



Notation

- Variables Y_i take values y_i
- Model output: $y = [y_1, \ldots, y_i, \ldots, y_n]$
- Cliques are sets of variable indices: $c = \{c_1, c_2, \ldots, c_k\}$
- Factors map partial variable assignments to real numbers: $\phi_c(y_{c_1}, \ldots, y_{c_k})$
- Probabilities are proportional to the product of all factors: $$P(y) \propto \prod_c \phi_c(y_{c_1}, \ldots, y_{c_k})$$

Abuse of Notation

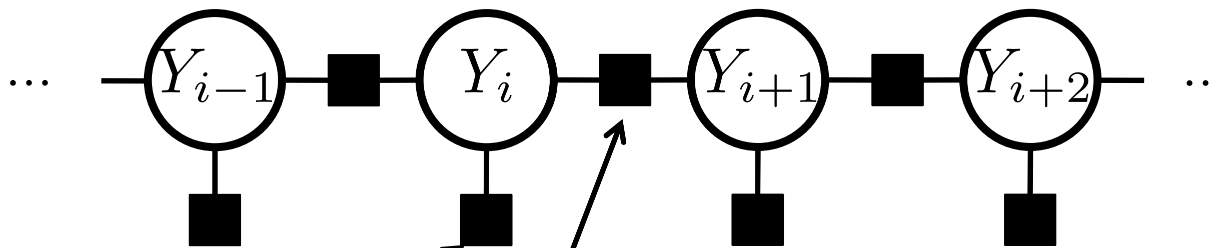
- For brevity, we will typically drop subscripts on potentials, for example writing $\phi(y_1, y_3, y_5)$ to mean:

$$\phi_{\{1,3,5\}}(Y_1 = y_1, Y_3 = y_3, Y_5 = y_5)$$

- It should always be clear from context which potential function is meant

Models: Examples

- Generative HMM (with x observed)



$$\phi(y_i) = p(x_i | y_i)$$

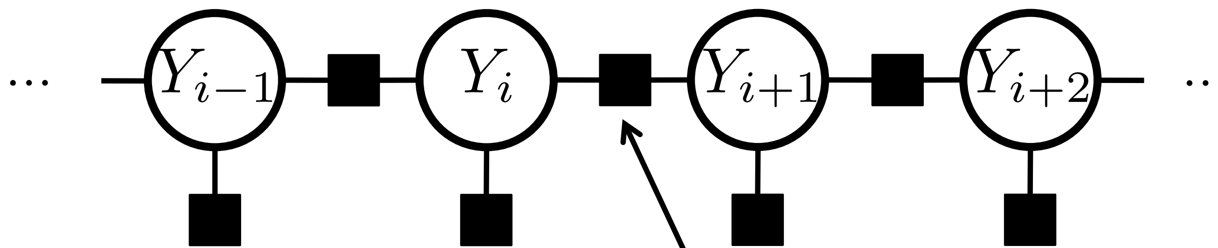
$$\phi(y_i, y_{i+1}) = p(y_{i+1} | y_i)$$

Observed variables do not appear in the graphical representation of the model, but are included in the relevant factors

Example application: computing posteriors for part-of-speech tagging, conditioned on sentence; each variable represents one tag

Models: Examples

- Exponential Family Chain CRF



Variable features: f_v
Edge features: f_e
Weight vector: w

$$\phi(y_i) = \exp\left(w^\top f_v(y_i)\right)$$

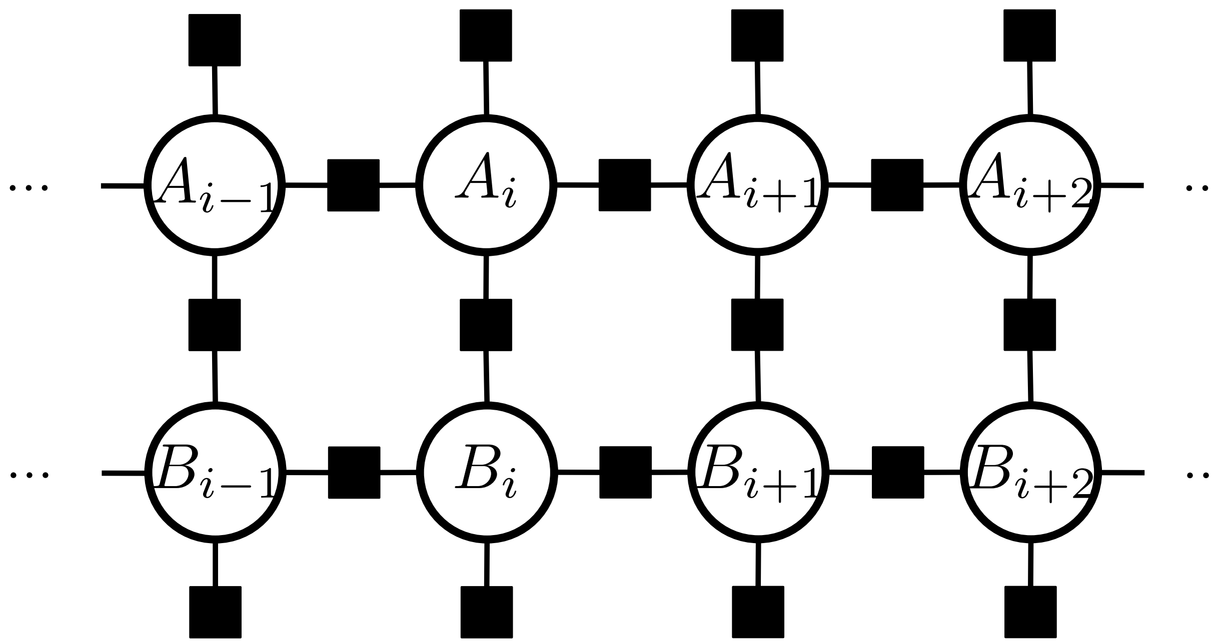
$$\phi(y_i, y_{i+1}) = \exp\left(w^\top f_e(y_i, y_{i+1})\right)$$

Example application: named entity recognition; each variable takes the value B(egin), I(nside), or O(utside)

[Lafferty et al, 2001]

Models: Examples

- Factorial Chain CRF



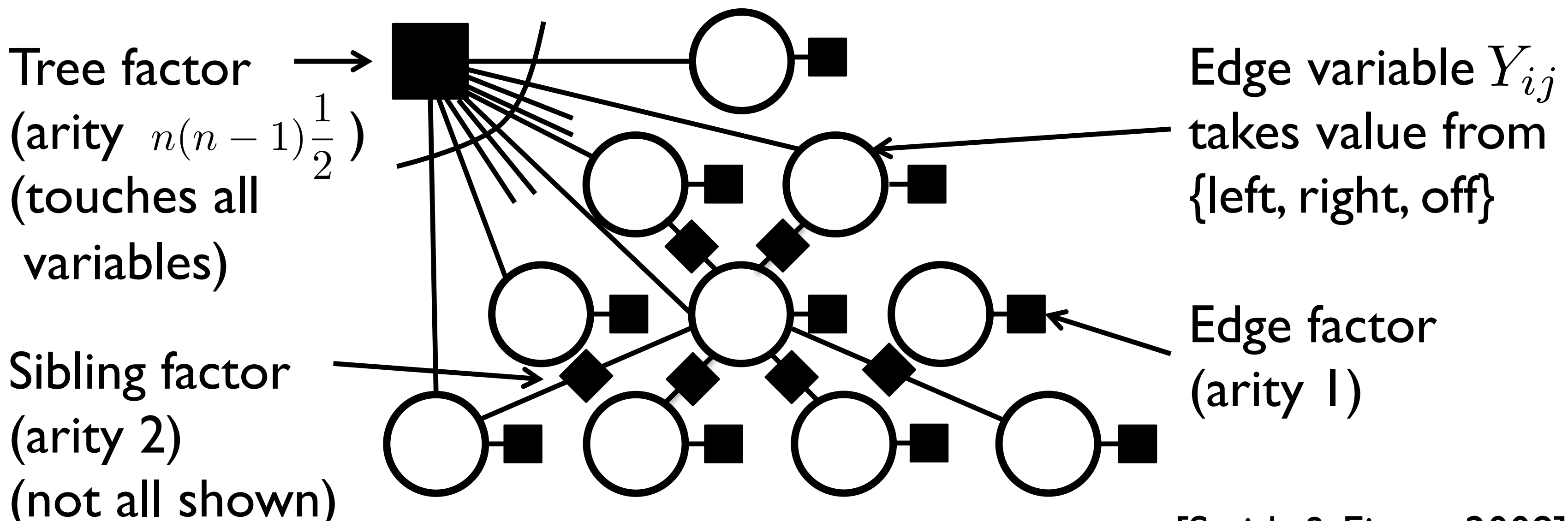
Example application: joint models of named entity recognition and part-of-speech tagging

[Sutton et al, 2004]

Models: Examples

- Dependency parsing with 2nd-order features (not all factors shown)

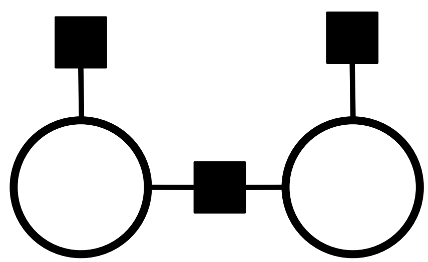
To encode the structural constraint on dependency parses, the tree factor is an indicator that has a value of 1 for well-formed trees and 0 otherwise



[Smith & Eisner, 2008]

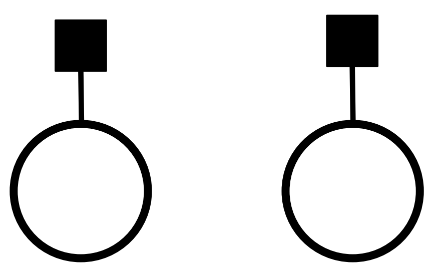
Mean Field Approximation

- General idea is to create an approximate distribution q whose parametric form is defined by a subgraph of the original graph
- Example:
  - Model:



$$p(y) \propto \phi(y_1)\phi(y_2)\phi(y_1, y_2)$$

  - Approximate Graph:



$$q(y) = q(y_1)q(y_2)$$

Mean Field Approximation

- Given the structure of q, the actual distribution is found by minimizing the KL divergence to p:

$$q = \operatorname*{argmin}_q KL(q||p)$$

$$= \operatorname*{argmin}_q \sum_y q(y) \log\left(\frac{q(y)}{p(y)}\right)$$

Mean Field Inference

- Mean field inference is iterative:
  1. Pick a component of $q(y)$ (e.g. $q(y_1)$)
  2. Find the distribution $q(y_1)$ that minimizes KL with other components of $q(y)$ fixed
  3. Repeat, cycling through components of $q(y)$ until convergence
- Finding the appropriate update in step 2 is the tricky part, so that is what we will focus our attention on

Mean Field Updates

- For naïve mean field, where the approximate graph contains only unary factors, the update rule is fairly simple:

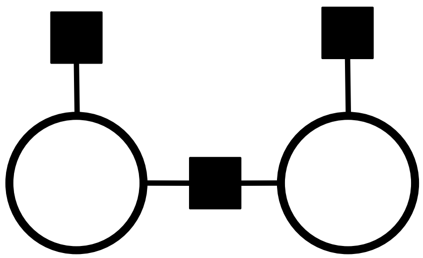
$$q(y_i) \propto \exp\left(\sum_{c:i\in c} \mathbb{E}_{q_{-Y_i}} \log \phi_c(y_c)\right)$$

- Here, $\mathbb{E}_{q_{-Y_i}}$ denotes the expectation with regard to all other components of q, so this expands to:

$$q(y_i) \propto \exp\left(\sum_{c:i\in c} \sum_{\substack{Y_c=y_c: \\ Y_i=y_i}} \left(\prod_{\substack{j\in c \\ j\neq i}} q(y_j)\right) \log \phi_c(y_c)\right)$$

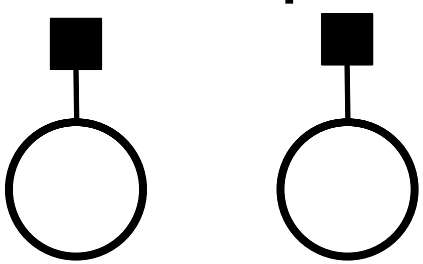
Mean Field Example Updates

Model:



$$p(y) \propto \phi(y_1)\phi(y_2)\phi(y_1, y_2)$$

Approximate Graph:



$$q(y) = q(y_1)q(y_2)$$

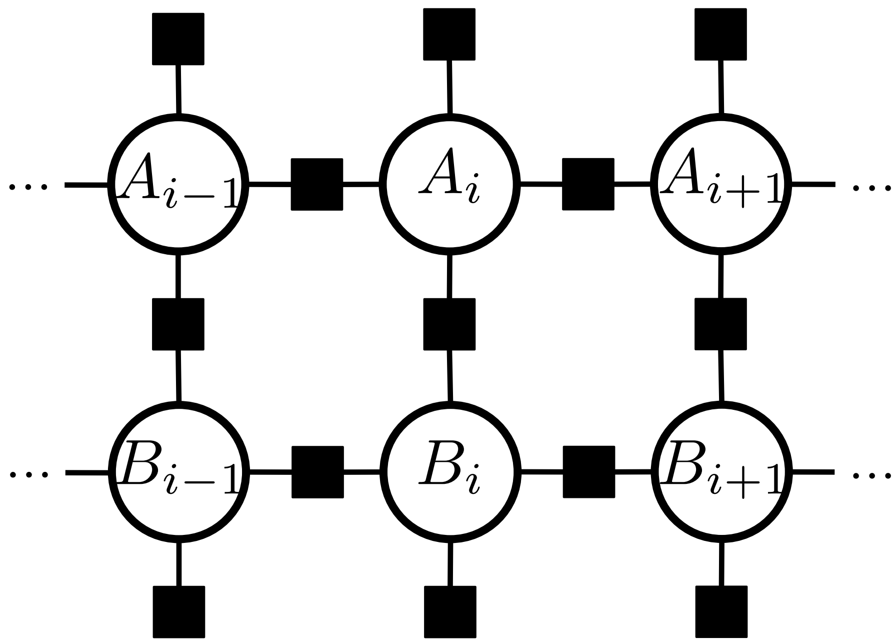
Update:

$$q(y_1) \propto \exp\left(\mathbb{E}_{q_2} \log(\phi(y_1)\phi(y_1, y_2))\right)$$

$$= \exp\left(\log \phi(y_1) + \sum_{y_2} q(y_2) \log(\phi(y_1, y_2))\right)$$

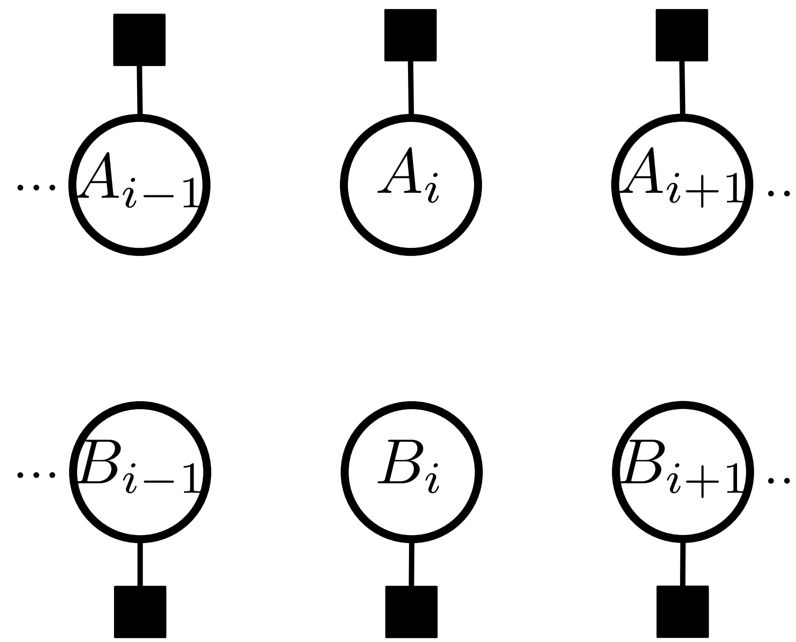
Mean Field Example Updates

Model:



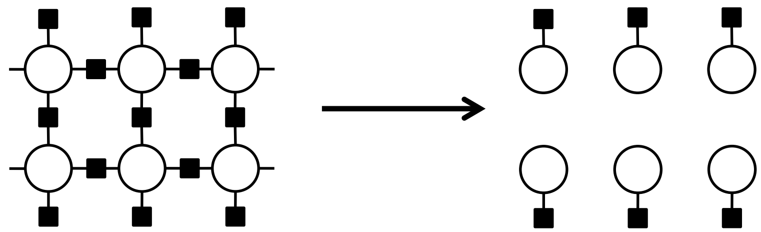
$$p(y) \propto \prod_i \phi(a_i)\phi(b_i)\phi(a_i, b_i) \phi(a_i, a_{i+1})\phi(b_i, b_{i+1})$$

Approximate Graph:



$$q(y) = \prod_i q(a_i)q(b_i)$$

Mean Field Example Updates



Update:

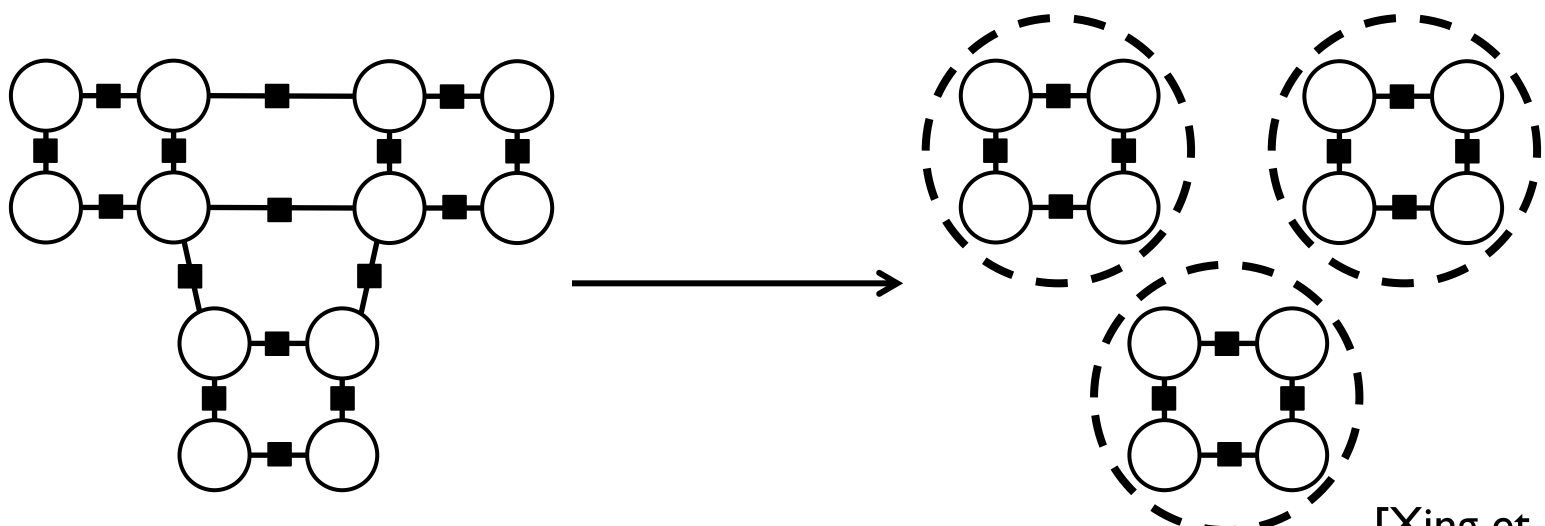
$$q(a_i) \propto \exp\left(\mathbb{E}_{q_{-A_i}} \log(\phi(a_i)\phi(a_{i-1},a_i)\phi(a_i,a_{i+1})\phi(a_i,b_i))\right)$$

$$= \exp\left(\log\phi(a_i) + \sum_{a_{i-1},a_{i+1},b_i} q(a_{i-1})q(a_{i+1})q(b_i)\log(\phi(a_{i-1},a_i)\phi(a_i,a_{i+1})\phi(a_i,b_i))\right)$$

$$= \exp\left(\log\phi(a_i) + \sum_{a_{i-1}} q(a_{i-1})\log(\phi(a_{i-1},a_i)) + \sum_{a_{i+1}} q(a_{i+1})\log(\phi(a_i,a_{i+1})) + \sum_{b_i} q(b_i)\log(\phi(a_i,b_i))\right)$$

Structured Mean Field

- In structured mean field, the subgraph defining the family of approximate distributions is selected by picking disjoint sets of variables to form connected components and removing all between-component links



[Xing et al, 2003]

Structured Mean Field

- Subgraphs are picked so that exact inference is possible within each individual connected component, either because the components are small or they have a dynamic program
- The mean field approximation then becomes a product of distributions over components
- Notation:
  - Connected components: $d = \{d_1, \ldots, d_m\}$
  - Approximation: $q(y) = \prod_d q(y_{d_1}, \ldots, y_{d_m})$

Component-Factorizable Factors

- There is a simple form for structured mean field updates, but its correctness is only guaranteed under some additional assumptions about the factor functions ϕ_c
- We say ϕ_c is *component-factorizable* if its log can be broken down to a product of per-component potentials: $\log \phi_c(y_c) = \prod_{d:c\cap d\neq\emptyset} \psi_{c\cap d}(y_{c\cap d})$
- The form we will give for structured mean field updates is correct if *all* factors are component-factorizable

Component-Factorizable Factors

- Note that while the component-factorizability restriction may seem onerous, it is typically achieved by the types of models common in the NLP literature
- In particular, exponential family models where $\phi_c(y_c) = \exp(w^\top f(y_c))$, with w a weight vector and $f(y_c)$ a vector of indicator features, usually satisfy this criterion (the overall indicator is typically a product of indicators that each component's requirement is satisfied)

Structured Mean Field Updates

- Instead of iterating through variables, we iterate over connected components, updating each $q(y_d)$ using marginals from $q(y_{-d})$:

$$q(y_d) \propto \exp\left(\sum_{c:c\cap d\neq\emptyset} \mathbb{E}_{q_{-d}} \log \phi(y_c)\right)$$

$$= \exp\left(\sum_{c:c\cap d\neq\emptyset} \sum_{\substack{Y_c=y_c:\\ Y_d=y_d}} \left(\prod_{d':c\cap d'\neq\emptyset} p_{q_{d'}}(y_{c\cap d'})\right) \log(\phi(y_c))\right)$$

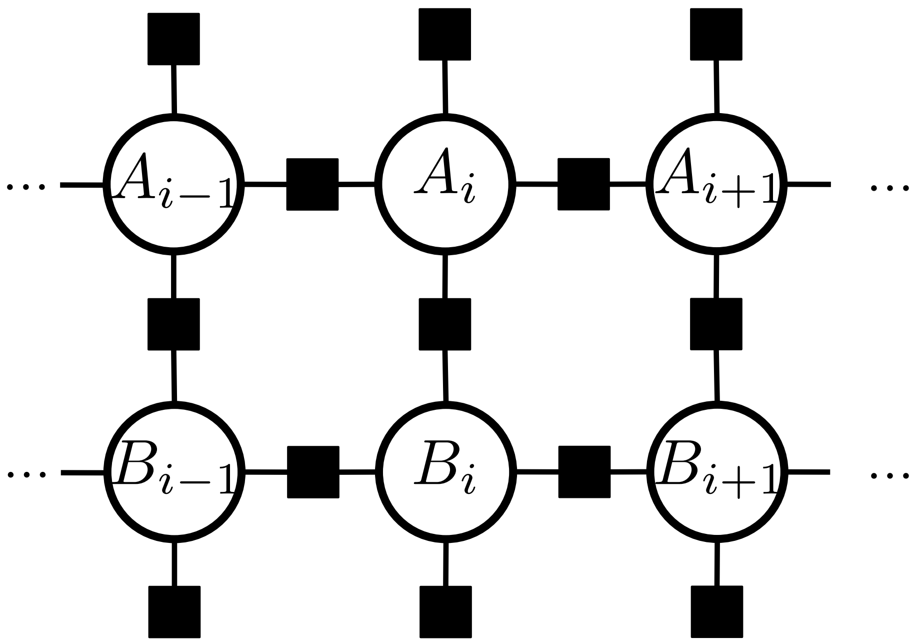
$p_{q_{d'}}(y_{c\cap d'})$ is the marginal probability of $y_{c\cap d'}$ under $q_{d'}$

Structured Mean Field Updates

- Note that the "updates" now define entire distributions q_d that are unlikely to ever be enumerated explicitly
- In practice, the iterative inference procedure consists of recomputing the marginal probabilities $p_{q_d}(y_{c \cap d})$ that appear in the definitions of the *other* components of $q(y)$

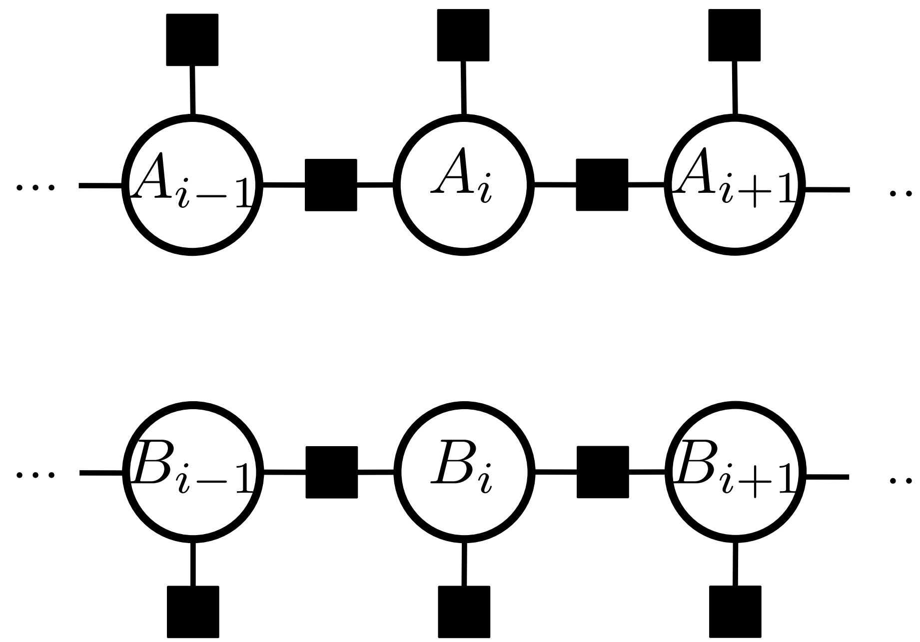
Structured Mean Field Example Update

Model:



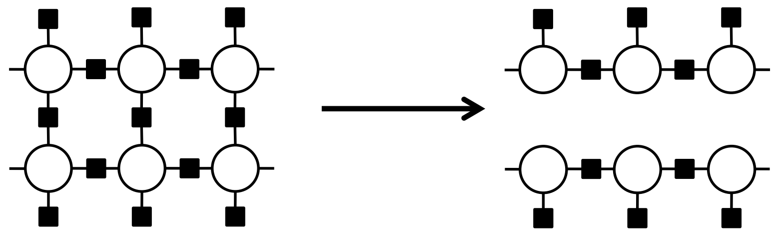
$$p(y) \propto \prod_i \phi(a_i)\phi(b_i)\phi(a_i, b_i)$$
$$\phi(a_i, a_{i+1})\phi(b_i, b_{i+1})$$

Approximation:



$$q(y) = q(a_{1,\ldots,n})q(b_{1,\ldots,n})$$

Structured Mean Field Example Update



Update:

$$q(a_{1,\ldots,n}) \propto \exp\left(\sum_i \mathbb{E}_{q_b} \log\left(\phi(a_i)\phi(a_i, a_{i+1})\phi(a_i, b_i)\right)\right)$$

$$= \exp\left(\sum_i \sum_b p_{q_b}(b_i) \log(\phi(a_i)\phi(a_i, a_{i+1})\phi(a_i, b_i))\right)$$

$$= \exp\left(\sum_i \log(\phi(a_i)) + \log(\phi(a_i, a_{i+1})) + \sum_{b_i} p_{q_b}(b_i) \log(\phi(a_i, b_i))\right)$$

Marginals $p_{q_b}(b_i)$ are computed using forward-backward on $q(b_{1,\ldots,n})$

Belief Propagation

- A message-passing algorithm used to compute approximate marginals
- Does not compute an entire approximate distribution – the "marginals" are not guaranteed to be consistent (i.e., marginals of any real distribution)
- Gives marginals on variables and on factors (marginals on ϕ_c are joint marginals on all the variables in c)

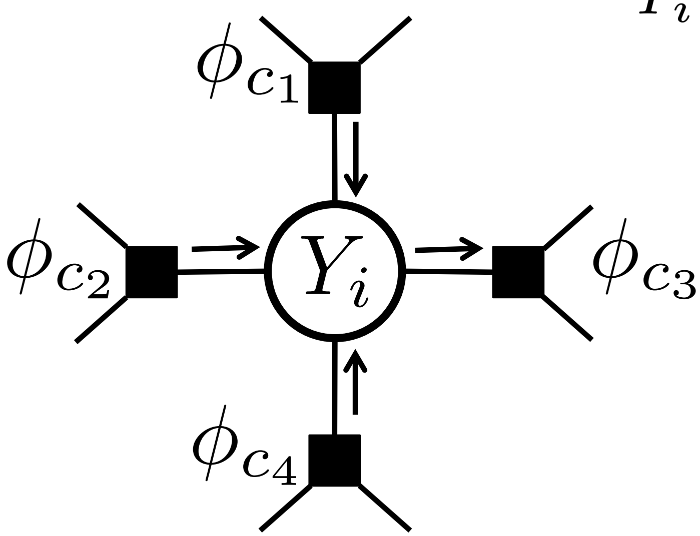
Messages

- There are two types of messages: from a variable to a neighboring factor, or from a factor to a neighboring variable
- All messages take the form of a (not necessarily normalized) distribution over the variable sending or receiving the message
- Messages at time $t+1$ are computed from messages at time t

Messages

- A message from variable Y_i to factor ϕ_c tells ϕ_c what Y_i thinks its own marginal distribution should be, based on the messages it has received from its *other* neighboring factors
- Example:

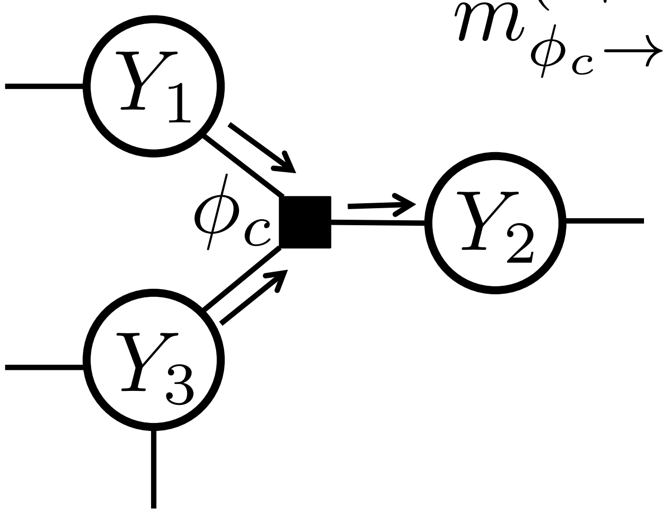
$$m^{(t+1)}_{Y_i \to \phi_{c_3}}(y_i) \propto m^{(t)}_{\phi_{c_1} \to Y_i}(y_i) \cdot m^{(t)}_{\phi_{c_2} \to Y_i}(y_i) \cdot m^{(t)}_{\phi_{c_4} \to Y_i}(y_i)$$



Messages

- A message from ϕ_c to Y_i tells what ϕ_c thinks Y_i's marginal should be based on ϕ_c's own factor function and the messages it has received from other variables in c
- Example:

$$m^{(t+1)}_{\phi_c \to Y_2}(y_2) \propto \sum_{y_1} \sum_{y_3} \phi_c(y_1, y_2, y_3) \cdot m^{(t)}_{Y_1 \to \phi_c}(y_1) \cdot m^{(t)}_{Y_3 \to \phi_c}(y_3)$$



Messages

- General form:
  - Messages from variables to factors:

$$m^{(t+1)}_{Y_i \to \phi_c}(y_i) \propto \prod_{\substack{c' \neq c: \\ Y_i \in c'}} m^{(t)}_{\phi_{c'} \to Y_i}(y_i) \qquad (1)$$

  - Messages from factors to variables:

$$m^{(t+1)}_{\phi_c \to Y_i}(y_i) \propto \sum_{\substack{Y_c = y_c: \\ Y_i = y_i}} \phi_c(y_c) \prod_{\substack{i' \neq i: \\ i' \in c}} m^{(t)}_{Y_{i'} \to \phi_c}(y_{i'}) \qquad (2)$$

Beliefs

- Marginal beliefs at time t are computed by just multiplying together all incoming messages
- Variable marginals:

$$b_{Y_i}^{(t)}(y_i) \propto \prod_{c:i\in c} m_{\phi_c \to Y_i}^{(t)}(y_i)$$

- Factor marginals:

$$b_{\phi_c}^{(t)}(y_c) \propto \phi_c(y_c) \prod_{i\in c} m_{Y_i \to \phi_c}^{(t)}(y_i)$$

Belief Propagation Algorithm

- Initialize all messages to some default (often uniform)
- Update each message according to equations (1) and (2)
- Repeat until convergence or until the maximum number of iterations is reached

Efficient Propagators

$$m^{(t+1)}_{\phi_c \to Y_i}(y_i) \propto \sum_{\substack{Y_c = y_c: \\ Y_i = y_i}} \phi_c(y_c) \prod_{\substack{i' \neq i: \\ i' \in c}} m^{(t)}_{Y_{i'} \to \phi_c}(y_{i'})$$

- Naïve computation of the sum in equation (2) takes time exponential in the arity of the factor
- For high-arity factors (e.g. the tree factor for dependency parsing), this means trouble
- Solution is to take advantage of sparsity (most variable assignments yield a factor value of 0) and the internal structure of the factor itself and use a dynamic program to compute all outgoing messages efficiently

Efficient Propagators

- A large arity factor ϕ_c starts with incoming messages $m_{Y_{c_1} \to \phi_c}, \ldots, m_{Y_{c_k} \to \phi_c}$ and needs to efficiently compute *all* outgoing messages: $m_{\phi_c \to Y_{c_1}}, \ldots, m_{\phi_c \to Y_{c_k}}$
- This can work if you have an appropriate dynamic program
- By initializing the dynamic program with values from the incoming messages, you can use it to compute marginal beliefs
- Outgoing messages are computed from marginal beliefs by dividing out the corresponding incoming message

Efficient Propagators

- Example: tree factor for dependency parsing
  - We need to add up scores for all legal dependency trees t
  - Denote the score of a single tree as $s(t)$; this is a product of $n-1$ incoming message values for "left" or "right", and the incoming message values of "off" for all the remaining edges (which do not appear in the tree)
  - The outgoing messages we need to compute have the form $m_{\phi_{\text{TREE}} \to Y_{ij}}(y_{ij}) = \sum_{t : y_{ij}(t) = y_{ij}} s(t)$, where $y_{ij}(t)$ has the value "left", "right", or "off", depending on whether, in t , i is the parent of j, j is the parent of i, or neither

[Smith & Eisner, 2008]

Efficient Propagators

- Example: tree factor for dependency parsing
  - Fortunately, dependency parsing algorithms, such as the Eisner algorithm, are really good at getting total scores for all trees with a particular edge: let's call these totals $\mu_{ij}(\text{left})$ and $\mu_{ij}(\text{right})$
  - We can also use the parsing algorithm to get a total score for all legal trees: we'll call this total Z
  - We're not quite there yet; total scores like $\mu_{ij}(\text{left})$ can't really incorporate messages from edges that don't appear
  - However, we can get around that by using the subsequent procedure, which scores edges with ratios instead of raw incoming message values

Efficient Propagators

- Example: tree factor for dependency parsing
  - First, compute $\pi = \prod_{ij} m_{Y_{ij} \to \phi_{\text{TREE}}}(\text{off})$
  - Initialize a parsing chart with the edge score for $Y_{ij} = \text{left}$ set to $\dfrac{m_{Y_{ij} \to \phi_{\text{TREE}}}(\text{ left })}{m_{Y_{ij} \to \phi_{\text{TREE}}}(\text{off})}$ (right is analogous)
  - Run the parser
  - Marginal beliefs are: $b_{ij}(\text{ left }) = \pi \mu_{ij}(\text{ left })$
  - Marginal off beliefs are computed by subtracting from Z: $b_{ij}(\text{off}) = \pi Z - (b_{ij}(\text{ left }) + b_{ij}(\text{ right }))$
  - Outgoing messages are computed by just dividing out incoming messages: $m_{\phi_{\text{TREE}} \to Y_i}(\text{ left }) \propto \dfrac{b_{ij}(\text{ left })}{m_{Y_i \to \phi_{\text{TREE}}}(\text{ left })}$

Efficient Propagators

- General procedure:
  - First, precompute the product of all incoming message scores for the "default" value
  - Initialize the chart of the dynamic program with odds ratios from incoming messages (non-default score divided by default score)
  - Run the dynamic program
  - Marginal beliefs for non-default values are the total scores from the dynamic program times the precomputed all-default score
  - Marginal default beliefs are taken by subtracting other marginal beliefs from the partition function

Belief Propagation Speed Tips

- Messages do not have to be updated in any particular order, but try to pick a schedule that tends to update messages after the ones they depend on
- You do not have to update each message every round; try doing multiple iterations of fast messages before updating slower ones (e.g. large structural factors)
- Messages for unary factors only have to be computed once

References

- John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira (2001). Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data. In ICML.
- David Smith and Jason Eisner (2008). Dependency Parsing by Belief Propagation. In EMNLP.
- Charles Sutton, Khashayar Rohanimanesh, and Andrew McCallum (2004). Dynamic Conditional Random Fields: Factorized Probabilistic Models for Labeling and Segmenting Sequence Data. In ICML.
- Eric P. Xing, Michael I. Jordan, and Stuart Russell (2003). A Generalized Mean Field Algorithm for Variational Inference in Exponential Families. In UAI.

(Full slides include a more thorough bibliography)