Joint Inference & FACTORIE 1.0

Andrew McCallum

Department of Computer Science
University of Massachusetts Amherst

Joint work with David Belanger, Sameer Singh, Alexandre Passos, Brian Martin, Michael Wick, Sebastian Riedel, and Limin Yao.
She saw the man with the telescope.
Natural Language Ambiguity

She saw the man with the telescope.

Parsing. Which PP attachment?

Stolen painting found by tree.

Semantic Role Labeling. Which semantic role?
Natural Language Processing

- Dialog
- Semantic Role Labeling
- Coreference Resolution
- Relation Extraction
- Entity Recognition
- Parsing
- Classification (POS, NER)
- Segmentation (word, phrase)

80-90% on many of these tasks.

But when combined errors cascade.

\[(0.9)^6 = 0.54\]
Unified Natural Language Processing

- Dialog
- Semantic Role Labeling
- Coreference Resolution
- Relation Extraction
- Entity Recognition
- Parsing
- Classification (POS, NER)
- Segmentation (word, phrase)

They’ll be no escape for the Princess this time.
Unified Computer Vision

Video Understanding
Object Tracking
Scene Understanding
Object Recognition
Object Segmentation
Depth Perception
Part Segmentation
Line Detection
Unified Robotics

Planning
Manipulation
Grasping
Object Recognition
State Estimation
Localization
Locomotion
Calibration
Joint Inference

Fundamental issue in all Artificial Intelligence.

(Learning is fundamental too, but more under control, I’d argue.)
That was inspirational.
But now for **Pessimism**

- Often a lot of trouble for moderate gains
  - Sutton & McCallum 2007, loopy BP, FCRF, 96.9% → 97.3%  +0.4%
  - Finkel & Manning 2008, forward sampling, 78.5% → 79.3%  +0.7%

- CoNLL 2009 Shared Task Summary:
  “The best systems overall do not use joint learning or optimization.”
Outline

• New algorithms for joint inference
  - Guaranteed-optimal “beam search” by Column Generation [NIPS 2012]
  - Joint inference with sparse values & factors [Submitted 2013]
  - Compressing messages with Expectation Propagation [TR 2012]
  - Focussed Query-specific MCMC Sampling [NIPS 2011]
  - Joint reasoning about relations of “universal schema” [AKBC 2012]

• **FACTORIE** Probabilistic programming support for joint inference
  - Landscape of tools
  - Capabilities
  - Release 1.0
Beam Search

• Message passing for finite-state inference with “spars-ified values” to be faster.

• Widely employed.

• Approximate. Non-optimal.

• **Our new work:**
  As fast as beam search, but *guaranteed* optimal.
  Answer always identical to exact Viterbi.
Inference in linear-chains
Inference in linear-chains
Inference in linear-chains

POS label

observed word

Time flies like an arrow
Inference in linear-chains

POS label
observed word

graphical model

Time flies like an arrow
Inference in linear-chains

POS label values
- DET
- NN
- VB
- ADJ
- ADV
- PP

graphical model
- POS label
- observed word
  - Time
  - flies
  - like
  - an
  - arrow
Inference in linear-chains

POS label values
- DET
- NN
- VB
- ADJ
- ADV
- PP

graphical model

POS label

observed word
- Time
- flies
- like
- an
- arrow
Inference in linear-chains
Inference in linear-chains

Beam Search

POS label values

DET
NN
VB
ADJ
ADV
PP

graphical model

POS label

observed word

Time
flies
like
an
arrow
Inference in linear-chains

Beam Search

POS label values

DET  NN  VB  ADJ  ADV  PP

graphical model

POS label

observed word

Time  flies  like  an  arrow
Inference in linear-chains

Beam Search

POS label values

- DET
- NN
- VB
- ADJ
- ADV
- PP

graphical model

observed word

Time
flies
like
an
arrow
Inference in linear-chains

Beam Search

POS label values

DET  NN  VB  ADJ  ADV  PP

graphical model

observed word

Time  flies  like  an  arrow
Inference in linear-chains

Beam Search

POS label values

- DET
- NN
- VB
- ADJ
- ADV
- PP

Graphical model

- POS label
- Observed word

- Time
- flies
- like
- an
- arrow
Inference in linear-chains
Message Passing with Column Generation
[Belanger, Passos, Riedel, McCallum, NIPS 2012]
Inference in linear-chains
Message Passing with Column Generation

[Belanger, Passos, Riedel, McCallum, NIPS 2012]

Related to Joshi’s “Bidirectional Incremental Construction”
Inference in linear-chains
Message Passing with Column Generation
[Belanger, Passos, Riedel, McCallum, NIPS 2012]

Related to Joshi’s “Bidirectional Incremental Construction”
Inference in linear-chains
Message Passing with Column Generation

[Belanger, Passos, Riedel, McCallum, NIPS 2012]

Related to Joshi’s “Bidirectional Incremental Construction”
Inference in linear-chains
Message Passing with Column Generation

[Belanger, Passos, Riedel, McCallum, NIPS 2012]

Related to Joshi’s “Bidirectional Incremental Construction”
Constrained Optimization

start with just subset of constraints

gradient direction = c
Constrained Optimization: *Cutting Planes*

find violated constraints, add just those
Constrained Optimization: **Cutting Planes**
find violated constraints, add just those

“Column Generation”: analogous idea for “values” instead of “constraints”

gradient direction = \( c \)
Linear Programming formulation of Viterbi (message passing in chains)

\[
\begin{align*}
\text{max.} & \quad \sum_{i, x_i, x_{i+1}} \mu_i(x_i, x_{i+1}) \left( \tau(x_i, x_{i+1}) + \frac{1}{2} \theta_i(x_i) + \frac{1}{2} \theta_{i+1}(x_{i+1}) \right) \\
\text{s.t.} & \quad \sum_{x_n} \mu_n(x_n, \cdot) = 1 \\
& \quad \sum_{x_1} \mu_0(\cdot, x_1) = 1 \\
& \quad \sum_{x_{i-1}} \mu_{i-1}(x_{i-1}, x_i) = \sum_{x_{i+1}} \mu_i(x_i, x_{i+1}) \\
& \quad \sum_{x_{i+1}} \mu_i(x_i, x_{i+1}) = \sum_{x_{i-1}} \mu_{i-1}(x_{i-1}, x_i)
\end{align*}
\]

New constraints:

\[
\begin{align*}
\alpha_{i+1}(x_{i+1}) &= \max_{x_i} \alpha_i(x_i) + \theta_i(x_i) + \tau(x_i, x_{i+1}) \\
\beta_{i-1}(x_{i-1}) &= \max_{x_i} \beta_i(x_i) + \theta_i(x_i) + \tau(x_{i-1}, x_i)
\end{align*}
\]
## Experiments

[Belanger, Passos, Riedel, McCallum, NIPS 2012]

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Speed</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Viterbi</td>
<td>1 sentences/s</td>
<td>100</td>
</tr>
<tr>
<td>Column generation</td>
<td>2.6 sentences/s</td>
<td>100</td>
</tr>
<tr>
<td>Beam-1</td>
<td>3.9 sentences/s</td>
<td>57.7</td>
</tr>
<tr>
<td>Beam-2</td>
<td>2.4 sentences/s</td>
<td>92.6</td>
</tr>
<tr>
<td>Beam-3</td>
<td>2.2 sentences/s</td>
<td>98.4</td>
</tr>
<tr>
<td>Beam-4</td>
<td>1.8 sentences/s</td>
<td>99.5</td>
</tr>
</tbody>
</table>

WSJ POS tagging, 45 labels
Synthetic Chains: real-world problems, described in the following sections.

To study the trade-offs provided by our approach, we run the inference algorithm on synthetic and real-world problems, described in the following sections.

4 Experiments

This approach can significantly improve performance even on a single core. Even though the total number of messages is still increasing the number of cores does not provide a benefit.

Although we describe this work on linear-chain models, the algorithm generalizes to the case of the tree. Exact inference in the tree is defined by selecting a root, and performing message passing in every stage of inference. However, we do not expect the speedups to be as significant as in chains.

Parallelism: The computations that are added to the queue are independent of each other, and thus can be computed simultaneously using multiple threads. For exact inference, the queue is initialized with only the variables as "islands of certainty" that each split the chain into two. This results in multiple forward passes and the end of forward pass to variable $f/b_i$ is set to the deterministic value if the mode of $m_{i-1} < m_i$ is different from $m_{i-1} < m_i$ and a pass to variable $f/b_i$.

Further, for every message computation, we do not add the number of messages to the queue of computations, i.e. linear speedups can be obtained if the queue size is always greater than the number of cores. However, the queue size doesn’t depend on the number of cores. Nonetheless, the queue size is not limited to splitting the tree into multiple smaller trees, thus providing potential for utilizing multiple cores at increasing the number of cores does not provide a benefit.

Marginals of a variable may vary considerably over the course of inference. After every message propagation, we update the deterministic value if the mode of $m_{i-1} < m_i$ is different from $m_{i-1} < m_i$. We also modify the deterministic value if the mode of $m_{i-1} < m_i$ is different from $m_{i-1} < m_i$. This approach can significantly improve performance even on a single core. Even though the total number of messages is still increasing the number of cores does not provide a benefit.

Figure 1: 4 threads 5x faster

Multi-core BP [Singh, Martin, McCallum, NIPS WS 2011]
## Experiments

[Belanger, Passos, Riedel, McCallum, NIPS 2012]

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Speed sentences/s</th>
<th>Accuracy % exact</th>
</tr>
</thead>
<tbody>
<tr>
<td>Viterbi</td>
<td>1</td>
<td>100</td>
</tr>
<tr>
<td>Column generation</td>
<td>2.6</td>
<td>100</td>
</tr>
<tr>
<td>Beam-1</td>
<td>3.9</td>
<td>57.7</td>
</tr>
<tr>
<td>Beam-2</td>
<td>2.4</td>
<td>92.6</td>
</tr>
<tr>
<td>Beam-3</td>
<td>2.2</td>
<td>98.4</td>
</tr>
<tr>
<td>Beam-4</td>
<td>1.8</td>
<td>99.5</td>
</tr>
</tbody>
</table>

WSJ POS tagging, 45 labels
Outline

• New algorithms for joint inference
  - Guaranteed-optimal “beam search” by Column Generation [NIPS 2012]
  - Joint inference with sparse values & factors [Submitted 2013]
  - Compressing messages with Expectation Propagation [TR 2012]
  - Focussed Query-specific MCMC Sampling [NIPS 2011]
  - Joint reasoning about relations of “universal schema” [AKBC 2012]

• FACTORIE  Probabilistic programming support for joint inference
  - Landscape of tools
  - Capabilities
  - Release 1.0
Schumacher has a contract with the Italian team through 2002.

Michael Schumacher is still celebrating Ferrari's ... "I'm still young."

Ferrari leads McLaren by 13 points.
we have only included 2.2, 2.3, and 2.4. See Figure 2 for an illustration of the joint model as defined over however, this factor induces a distribution over both the pairwise boolean variable and the pairwise boolean coreference variable given the entity tags of the mentions. In the joint model, now directly represent the

Formally, the probability for a setting to all the variables in a document is:

\[ p(t, r, c|m) \propto \prod_{t_i \in t} \Psi_t(m_i, t_i) \prod_{c_{ij} \in c} \Psi_c(c_{ij}, m_i, m_j, t_i, t_j) \prod_{r_{ij} \in r} \Psi_r(r_{ij}, m_i, m_j, t_i, t_j) \]

underlying joint model is quite complex and dense. Variable counts over the train, test, and two of which are in the same sentence. Note that even with such a small set of mentions, the models are also the set of factors instantiated by individual models, as described in sections

there is no direct linkage between relation extraction and coreference. Note that for brevity, each coreference factor defines a distribution over the labels of a single task conditioned on the predictions from another task, these factors

relations between mentions

Inference with dynamically spars-ified values and factors.
Experimental Results

Number of variables of each type

<table>
<thead>
<tr>
<th>Source</th>
<th>#Mentions</th>
<th>#Coreference</th>
<th>#Relation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Train</td>
<td>15,640</td>
<td>637,160</td>
<td>82,479</td>
</tr>
<tr>
<td>Dev</td>
<td>5,545</td>
<td>244,461</td>
<td>34,057</td>
</tr>
<tr>
<td>Test</td>
<td>6,598</td>
<td>342,942</td>
<td>38,270</td>
</tr>
</tbody>
</table>

Our approach to joint inference differs significantly from those above. First, we are modeling three crucial information extraction tasks, including coreference, which have not been modeled together before. Coreference as the third task requires document-level joint inference, as opposed to sentence-level joint inference in related work. Difficulty of inference is further compounded from transitivity. Second, our resulting model is significantly more loopy than a number of existing joint inference techniques. Due to its size and structure, most approximate inference techniques are not directly applicable, and we present a novel efficient inference technique that is a modification of belief propagation. Third, as opposed to some of the related work, we learn both hard and soft constraints between tasks instead of setting them by hand (as in Roth and Yih (2007)), and our inference provides marginals. Due to the dependencies represented in our model, and our inference technique, we are able to obtain consistent improvements in all the three tasks, improving accuracy as we include more dependencies.

Experiments

We use the Automatic Content Extraction (ACE; Doddington et al. (2004)) 2004 English dataset for the experiments, a standard labeled corpus for the three tasks that we are studying. ACE consists of 443 documents from 4 distinct news domains: broadcast, newswire, Arabic treebank, and Chinese treebank. The total number of sentences and tokens are 7,790 and 172,506 respectively. We split the data into train, test, and development sets of sizes 60%, 20%, and 20%. Counts of each type of variable are shown in Table 5.

Our model takes the complete mention string as input, and for these experiments we use the gold mention boundaries.

Due to the complexity of inference and the size and density of the model, we restrict the set of labels we explore for both entity tags and relation extraction to the coarse-grained types (7 and 8 respectively). Using predicted mention boundaries remains future work, although we imagine boundary detection can be yet another task for joint inference.

<table>
<thead>
<tr>
<th></th>
<th>Entity Tagging</th>
<th>Relation Extraction</th>
<th>Entity Resolution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pipeline</td>
<td>80%</td>
<td>54%</td>
<td>58%</td>
</tr>
<tr>
<td>Joint</td>
<td>83%</td>
<td>55%</td>
<td>79%</td>
</tr>
</tbody>
</table>

50% reduction in error!
Outline

• New algorithms for joint inference
  - Guaranteed-optimal “beam search” by Column Generation [NIPS 2012]
  - Joint inference with sparse values & factors [Submitted 2013]
  - Compressing messages with Expectation Propagation [TR 2012]
  - Focussed Query-specific MCMC Sampling [NIPS 2011]
  - Joint reasoning about relations of “universal schema” [AKBC 2012]

• FACTORIE Probabilistic programming support for joint inference
  - Landscape of tools
  - Capabilities
  - Release 1.0
Using 2.2.1 Sparse and Merged Messages

The upstream pass is performed using the update rule:

\( d(t) \leftarrow \mu(t) \cdot \prod_{i \in \mathcal{D}(t)} \frac{Z_i(x)}{Z_i} \)

\( \tilde{Z}(u) = \prod_{i \in \mathcal{D}(u)} Z_i(x) \)

then we estimate a distribution over the joint model using EP, this means that instead of matching global statistics, we estimate and seek to match per-variable expectations exactly, instead they assign zero ability so that all the remaining mass equally among the states of some small set of states.

2.2.2 Belief Propagation

Factorial chain between two modules can also sample the list. For the set of \( \mathcal{L} \) constraints, we can estimate the uncertainty of the sparse message approaches more likely than the sparse message approaches, like

\( \mathcal{L} \cdot \mathcal{D} = \mathcal{L} \cdot \mathcal{D} \cdot \mathcal{X} \)

Coordinate by passing sparse messages among the parts.

Cut complex graph into easier-to-manage parts.

Coordinate by passing messages among the parts.
BP with EP Message Compression

[Singh, Riedel, McCallum, 2012]
BP with EP Message Compression
[Singh, Riedel, McCallum, 2012]

Compress by using a graphical model to represent the message.

Results on Joint Seg & Coref of citation data

[Singh, Riedel, McCallum, 2012]

<table>
<thead>
<tr>
<th>Method</th>
<th>Prec</th>
<th>Rec</th>
<th>F1</th>
<th>Time (hours)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k$-Best</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$k = 1$</td>
<td>98.8</td>
<td>42.3</td>
<td>59.3</td>
<td>0.08</td>
</tr>
<tr>
<td>$k = 2$</td>
<td>98.9</td>
<td>40.6</td>
<td>57.6</td>
<td>0.16</td>
</tr>
<tr>
<td>$k = 5$</td>
<td>99.1</td>
<td>46.5</td>
<td>63.3</td>
<td>0.4</td>
</tr>
<tr>
<td>$k = 10$</td>
<td>98.8</td>
<td>48.1</td>
<td>64.8</td>
<td>0.8</td>
</tr>
<tr>
<td>$k = 50$</td>
<td>98.2</td>
<td>53.6</td>
<td>69.3</td>
<td>4.0</td>
</tr>
<tr>
<td>$k = 100$</td>
<td>98.2</td>
<td>54.3</td>
<td>69.9</td>
<td>8.0</td>
</tr>
<tr>
<td>$k = 150$</td>
<td>98.2</td>
<td>54.8</td>
<td>70.3</td>
<td>12.0</td>
</tr>
<tr>
<td>Sampling</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$s = 25$</td>
<td>96.3</td>
<td>67.6</td>
<td>79.4</td>
<td>3.1</td>
</tr>
<tr>
<td>$s = 100$</td>
<td>96.9</td>
<td>67.3</td>
<td>79.5</td>
<td>10.0</td>
</tr>
<tr>
<td>Our EP method</td>
<td>95.3</td>
<td>69.9</td>
<td>80.7</td>
<td>0.16</td>
</tr>
</tbody>
</table>
Outline

• **New algorithms for joint inference**
  - Guaranteed-optimal “beam search” by Column Generation [NIPS 2012]
  - Joint inference with sparse values & factors [Submitted 2013]
  - Compressing messages with Expectation Propagation [TR 2012]
  - **Focussed Query-specific MCMC Sampling** [NIPS 2011]
  - Joint reasoning about relations of “universal schema” [AKBC 2012]

• **FACTORIE** Probabilistic programming support for joint inference
  - Landscape of tools
  - Capabilities
  - Release 1.0
Query-specific Sampling

Given a query, no need to run sampling on whole universe of data

Want *query-specific sampling*. (Amazingly: under-studied problem!)
Query-specific Sampling

[Wick, McCallum, NIPS 2011]

- Define *influence trail score* as approximation to marginal MI.
- Prove Proposition 1. If \( p(i) = 1(i \neq l) \frac{1}{n-1} \) induces an MH kernel that neglects variable \( x_l \), then the expected total variation error \( \xi_{tv} \) of the resulting MH sampling procedure under the model is the total variation influence \( \iota_{tv} \).

\[
\iota_{tv}(Q) = \mathbb{E}_{x_{Q}, y_{\tilde{Q}} \sim q} \left[ d_{tv}(\rho(Q), \rho_{\tilde{Q}}) \right]
\]

\[d_{tv}(\rho_1, \rho_2) = \int |\rho_1(x) - \rho_2(x)| dx\]

\[\rho_{\tilde{Q}}(x) = \frac{1}{n-1} \sum_{l \neq l} \rho(x)
\]

\[\rho_{\tilde{Q}}(x) = \frac{1}{n-1} \sum_{l \neq l} \rho(x)
\]

\[\iota_{tv}(Q) = \mathbb{E}_{x_{Q}, y_{\tilde{Q}} \sim q} \left[ d_{tv}(\rho(Q), \rho_{\tilde{Q}}) \right]
\]

\[d_{tv}(\rho_1, \rho_2) = \int |\rho_1(x) - \rho_2(x)| dx\]

\[\rho_{\tilde{Q}}(x) = \frac{1}{n-1} \sum_{l \neq l} \rho(x)
\]

\[\rho_{\tilde{Q}}(x) = \frac{1}{n-1} \sum_{l \neq l} \rho(x)
\]

\[\iota_{tv}(Q) = \mathbb{E}_{x_{Q}, y_{\tilde{Q}} \sim q} \left[ d_{tv}(\rho(Q), \rho_{\tilde{Q}}) \right]
\]

\[d_{tv}(\rho_1, \rho_2) = \int |\rho_1(x) - \rho_2(x)| dx\]

\[\rho_{\tilde{Q}}(x) = \frac{1}{n-1} \sum_{l \neq l} \rho(x)
\]

\[\rho_{\tilde{Q}}(x) = \frac{1}{n-1} \sum_{l \neq l} \rho(x)
\]

\[\iota_{tv}(Q) = \mathbb{E}_{x_{Q}, y_{\tilde{Q}} \sim q} \left[ d_{tv}(\rho(Q), \rho_{\tilde{Q}}) \right]
\]

\[d_{tv}(\rho_1, \rho_2) = \int |\rho_1(x) - \rho_2(x)| dx\]

\[\rho_{\tilde{Q}}(x) = \frac{1}{n-1} \sum_{l \neq l} \rho(x)
\]

\[\rho_{\tilde{Q}}(x) = \frac{1}{n-1} \sum_{l \neq l} \rho(x)
\]

\[\iota_{tv}(Q) = \mathbb{E}_{x_{Q}, y_{\tilde{Q}} \sim q} \left[ d_{tv}(\rho(Q), \rho_{\tilde{Q}}) \right]
\]

\[d_{tv}(\rho_1, \rho_2) = \int |\rho_1(x) - \rho_2(x)| dx\]
Outline

• **New algorithms for joint inference**
  - Guaranteed-optimal “beam search” by Column Generation [NIPS 2012]
  - Joint inference with sparse values & factors [Submitted 2013]
  - Compressing messages with Expectation Propagation [TR 2012]
  - Focussed Query-specific MCMC Sampling [NIPS 2011]
  - Joint reasoning about relations of “universal schema” [AKBC 2012]

• **FACTORIE** Probabilistic programming support for joint inference
  - Landscape of tools
  - Capabilities
  - Release 1.0
Outline

• New algorithms for joint inference
  - Guaranteed-optimal “beam search” by Column Generation [NIPS 2012]
  - Joint inference with sparse values & factors [Submitted 2013]
  - Compressing messages with Expectation Propagation [TR 2012]
  - Focussed Query-specific MCMC Sampling [NIPS 2011]
  - Joint reasoning about relations of “universal schema” [AKBC 2012]

• FACTORIE Probabilistic programming support for joint inference
  - Landscape of tools
  - Capabilities
  - Release 1.0
Really Hairy Models!
How to do

- parameter estimation
- inference
Joint Inference

Really Hairy Models!

How to do

- parameter estimation
- inference
- software engineering
Probabilistic Programming Languages

• Make it easy to specify rich, complex models, using the full power of programming languages
  - data structures
  - control mechanisms
  - abstraction

• Inference implementation comes for free

Provides language to easily create new models
Landscape of Tools

• **BUGS** Bayesian Inf. using Gibbs Sampling
  - Arbitrary graphical models
  - Slow, small data

• **Alchemy** Markov Logic [Domingos]
  - Convenient language
  - Unpredictable inefficiency

• **GraphLab** [Guestrin]
  - Scalable parallel/distributed
  - Knows nothing of prob. model

• **Stanford NLP Toolkit** [Manning]
  - Rich NLP data types & models
  - Ad-hoc collection of implementations
“Factor Graphs, Imperative, Extensible”

Implemented as a library in Scala [Martin Odersky]
- object oriented & functional; type inference
- runs in JVM (complete interoperation with Java)
- fast, JIT compiled, but also cmd-line interpreter

Library, not new “little language”
- integrate data pre-processing & eval. w/ model spec
- leverage OO-design: modularity, encapsulation, inheritance

Distinct: Data, Factors, Inference, Learning

Pre-build, rich NLP representations & algorithms

Scalable
- billions of variables, super-exponential #factors
- DB back-end
Stages of FACTORIE programming

1. Define “templates for data” (i.e. classes of random variables)
   - Use data structures just like in deterministic programming.
   - Easily express relations among pieces of data.

2. Define “templates for factors” (i.e. model)
   - Distinct from above data representation; makes it easy to modify model scoring independently.
   - Define factor scores based on neighboring variable values.

3. Select inference (message-passing, sampling,...)
   - Gibbs Sampling, Metropolis-Hastings, Iterated Conditional Modes, WalkSAT, LP, Belief propagation, SparseBP, Expectation Propagation,...

4. Select learning (loss function, optimization, parallelization strategy)

5. Read the data, creating variables.
   Then inference / parameter estimation is often a one-liner!
FACTORIE “special sauce”

• Other tools (e.g. MSR’s Infer.NET) are very powerful but inference is black box. Penetrable only by architects

• FACTORIE is designed to make it easy for users to descend layers of abstraction.

  e.g.
  1. Command-line tool
  2. Short script with pre-selected variables, model, inference, learning
  3. Longer script with custom selections
  4. Implement custom data structures new variables
  5. Implement custom components inference, learning
Components / Vocabulary

- **Variable**  value, set, domain
- **Domain**  values
- **Diff, DiffList**  redo, undo
- **Factor**  variables, score
- **Family (of factors)**  weights
- **Template (of factors)**  factors, unroll1, unroll2
- **Model (& objective)**  factors, score
- **Marginal, Summary**  marginal(variables)
- **Inferencer**  infer(variables): Summary
- **Optimizer**  step(weights, gradient, value)
- **Piece (of data)**  accumulateGradient(model, accumulator)
- **Trainer**  model, process(pieces)
Example Variables

• RealVariable
• IntegerVariable
• DiscreteVariable, CategoricalVariable
• TensorVariable
• ProportionsVariable, MassesVariable
• SeqVariable
• EdgeVariable
• SetVariable
• RefVariable
Example NLP Variables

• Token, Span, Sentence
  - token.attr[PosLabel], span.attr[NerLabel]
• Sentence
• Document
• Mention, Entity
• Relation
• ParseTree
Example NLP Components

- Token segmentation
- Sentence segmentation
- Part-of-speech tagging
- Noun phrase chunking
- Named-entity extraction
- Dependency parsing
- Relation extraction
- Within-doc and cross-doc coreference
Example: Binary Doc Classifier

// Data
class Document extends FeatureVectorVariable { def domain = DocumentDomain }
class Label(val doc: Document) extends LabeledBooleanVariable
Example: Binary Doc Classifier

// Data
class Document extends FeatureVectorVariable {
  def domain = DocumentDomain
}
class Label(val doc: Document) extends LabeledBooleanVariable
val data: Iterable[Label] = readData()
Example: Binary Doc Classifier

// Data
class Document extends FeatureVectorVariable {
    def domain = DocumentDomain
}
class Label(val doc:Document) extends LabeledBooleanVariable
val data: Iterable[Label] = readData()

// Model
var model = new Model[Label] {

}
Example: Binary Doc Classifier

// Data
class Document extends FeatureVectorVariable {
  def domain = DocumentDomain
}
class Label(val doc: Document) extends LabeledBooleanVariable
val data: Iterable[Label] = readData()

// Model
var model = new Model[Label] {
  val bias = new DotFamilyWithStatistics1[Label] {
    val weights = new 
  }
  val obs = new DotFamilyWithStatistics[Label, Document] {
    val weights = 
  }
}
Example: Binary Doc Classifier

// Data
class Document extends FeatureVectorVariable { def domain = DocumentDomain }
class Label(val doc:Document) extends LabeledBooleanVariable
val data: Iterable[Label] = readData()

// Model
var model = new Model[Label] {
  val bias = new DotFamilyWithStatistics1[Label] { val weights = new
  val obs = new DotFamilyWithStatistics[Label,Document] { val weights =
  def factors(label:Label) = List(
    bias.Factor(label), obs.Factor(label, label.token) )
}
Example: Binary Doc Classifier

traditional batch training, L-BFGS updates

// Data
class Document extends FeatureVectorVariable { def domain = DocumentDomain }
class Label(val doc:Document) extends LabeledBooleanVariable
val data: Iterable[Label] = readData()

// Model
var model = new Model[Label] {
  val bias = new DotFamilyWithStatistics1[Label] { val weights = new
  val obs = new DotFamilyWithStatistics[Label,Document] { val weights =
    def factors(label:Label) = List(
      bias.Factor(label), obs.Factor(label, label.token) )
  }
}

// Learning
val trainer = new BatchTrainer(L2RegularizedLBFGS, model)
trainer.process(data.map(y => new GLMPiece(y)))
Example: Binary Doc Classifier

traditional batch training, L-BFGS updates

```scala
// Data
class Document extends FeatureVectorVariable { def domain = DocumentDomain }
class Label(val doc: Document) extends LabeledBooleanVariable
val data: Iterable[Label] = readData()

// Model
var model = new Model[Label] {
  val bias = new DotFamilyWithStatistics1[Label] { val weights = new
  val obs = new DotFamilyWithStatistics[Label, Document] { val weights =
  def factors(label: Label) = List(
    bias.Factor(label), obs.Factor(label, label.token) )
}

// Learning
val trainer = new BatchTrainer(L2RegularizedLBFGS, model)
trainer.process(data.map(y => new GLMPiece(y))

// Inference
val marginal = Infer(data.first, model)
```
Example: Binary Doc Classifier
stochastic gradient ascent, plain gradient updates

// Data
class Document extends FeatureVectorVariable {
  def domain = DocumentDomain
}
class Label(val doc:Document) extends LabeledBooleanVariable
val data: Iterable[Label] = readData()

// Model
var model = new Model[Label] {
  val bias = new DotFamilyWithStatistics1[Label] {
    val weights = new 
    val obs = new DotFamilyWithStatistics[Label,Document] {
      val weights = 
      def factors(label:Label) = List(
        bias.Factor(label), obs.Factor(label, label.token) )
    }
  }
}

// Learning
val trainer = new SGDTrainer(StepwiseGradientAscent, model)
trainer.process(data.map(y => new GLMPiece(y))

// Inference
val marginal = Infer(data.first, model)
Example: Binary Doc Classifier

stochastic gradient ascent, with *conf. weighting* [Pereira]

```scala
// Data
class Document extends FeatureVectorVariable {
  def domain = DocumentDomain
}
class Label(val doc: Document) extends LabeledBooleanVariable
val data: Iterable[Label] = readData()

// Model
var model = new Model[Label] {
  val bias = new DotFamilyWithStatistics1[Label] {
    val weights = new
    val obs = new DotFamilyWithStatistics[Label, Document] {
      val weights =
      def factors(label: Label) = List(
        bias.Factor(label), obs.Factor(label, label.token)
      )
    }
  }
}

// Learning
val trainer = new SGDTrainer(AROW, model)
trainer.process(data.map(y => new GLMPiece(y))

// Inference
val marginal = Infer(data.first, model)
```
Example: Binary Doc Classifier
“hogwild” parallelized stochastic gradient ascent

```
// Data
class Document extends FeatureVectorVariable { def domain = DocumentDomain }
class Label(val doc:Document) extends LabeledBooleanVariable
val data: Iterable[Label] = readData()

// Model
var model = new Model[Label] {
  val bias = new DotFamilyWithStatistics1[Label] { val weights = new 
  val obs = new DotFamilyWithStatistics[Label,Document] { val weights =
  def factors(label:Label) = List(
    bias.Factor(label), obs.Factor(label, label.token) )
}

// Learning
val trainer = new HogwildTrainer(AROW, model)
trainer.process(data.map(y => new GLMPiece(y))

// Inference
val marginal = Infer(data.first, model)
```
Example: **N-ary Doc Classifier**

“hogwild” parallelized stochastic gradient ascent

```scala
// Data
class Document extends FeatureVectorVariable { def domain = DocumentDomain }
class Label(val doc:Document) extends LabeledCategoricalVariable { def domain = DocumentDomain }
val data: Iterable[Label] = readData()

// Model
var model = new Model[Label] {
  val bias = new DotFamilyWithStatistics1[Label] { val weights = new
  val obs = new DotFamilyWithStatistics[Label,Document] { val weights =
  def factors(label:Label) = List(
    bias.Factor(label), obs.Factor(label, label.token) )
}

// Learning
val trainer = new HogwildTrainer(AROW, model)
trainer.process(data.map(y => new GLMPiece(y))

// Inference
val marginal = Infer(data.first, model)
```
Example: N-ary Doc Classifier

“hogwild” with SampleRank gradients

// Data
class Document extends FeatureVectorVariable { def domain = DocumentDomain } class Label(val doc:Document) extends LabeledCategoricalVariable { def val data: Iterable[Label] = readData() } // Model
var model = new Model[Label] {
  val bias = new DotFamilyWithStatistics1[Label] { val weights = new
  val obs = new DotFamilyWithStatistics[Label,Document] { val weights =
  def factors(label:Label) = List(
    bias.Factor(label), obs.Factor(label, label.token) )
}
// Learning
val trainer = new HogwildTrainer(AROW, model)
trainer.process(data.map(y => new SampleRankPiece(y))
// Inference
val marginal = Infer(data.first, model)
Example: **Sequence Labeling**

“hogwild” with SampleRank gradients

```scala
// Data
class Token extends FeatureVectorVariable { def domain = DocumentDomain }
class Label(val doc: Document) extends LabeledCategoricalVariable { def
val sentences: Iterable[Seq[Label]] = readData()

// Model
var model = new Model[Label] {
  val bias = new DotFamilyWithStatistics1[Label] { val weights = new
  val obs = new DotFamilyWithStatistics[Label, Document] { val weights =
  def factors(label: Label) = List(
    bias.Factor(label), obs.Factor(label, label.token) )
}

// Learning
val trainer = new HogwildTrainer(AROW, model)
trainer.process(sentences.map(y => new SampleRankPiece(y))

// Inference
val marginals = Infer(sentences.first, model)
```
Example: Sequence Labeling
“hogwild” with SampleRank gradients, with Markov deps

// Data
class Token extends FeatureVectorVariable { def domain = DocumentDomain }
class Label(val doc:Document) extends LabeledCategoricalVariable { def val sentences: Iterable[Seq[Label]] = readData()

// Model
var model = new Model[Seq[Label]] {
    val bias = new DotFamilyWithStatistics1[Label] { val weights = new 
    val obs = new DotFamilyWithStatistics[Label,Document] { val weights = 
    val markov = new DotFamilyWithStatistics2[Label,Label] { val weights 
    def factors(labels:Seq[Label]) =
        for (label <- labels) yield bias.Factor(label) ++ 
        for (label <- labels) yield obs.Factor(label, label.token) ++ 
        for (label <- labels.drop(1)) yield markov.Factor(label.prev, label
    }

// Learning
val trainer = new HogwildTrainer(AROW, model)
trainer.process(sentences.map(y => new SampleRankPiece(y))

// Inference
val marginals = Infer(sentences.first, model)
Example: Sequence Labeling
“hogwild”..., with Markov deps, Gibbs Sampling inference

// Data
class Token extends FeatureVectorVariable { def domain = DocumentDomain }
class Label(val doc:Document) extends LabeledCategoricalVariable { def 
val sentences: Iterable[Seq[Label]] = readData()

// Model
var model = new Model[Seq[Label]] {
  val bias = new DotFamilyWithStatistics1[Label] { val weights = new 
  val obs = new DotFamilyWithStatistics[Label,Document] { val weights = 
  val markov = new DotFamilyWithStatistics2[Label,Label] { val weights 
  def factors(labels:Seq[Label]) =
    for (label <- labels) yield bias.Factor(label) ++
    for (label <- labels) yield obs.Factor(label, label.token) ++
    for (label <- labels.drop(1)) yield markov.Factor(label.prev, label

// Learning
val trainer = new HogwildTrainer(AROW, model)
trainer.process(sentences.map(y => new SampleRankPiece(y))

// Inference
val marginals = GibbsSampling.infer(sentences.first, model)
Example: Sequence Labeling
“hogwild”..., with Markov deps, BP inference

// Data
class Token extends FeatureVectorVariable {
  def domain = DocumentDomain
}
class Label(val doc:Document) extends LabeledCategoricalVariable {
  def domain = DocumentDomain
}
val sentences: Iterable[Seq[Label]] = readData()

// Model
var model = new Model[Seq[Label]] {
  val bias = new DotFamilyWithStatistics1[Label] {
    val weights = new
  }
  val obs = new DotFamilyWithStatistics[Label,Document] {
    val weights = new
  }
  val markov = new DotFamilyWithStatistics2[Label,Label] {
    val weights = new
  }
  def factors(labels:Seq[Label]) = {
    for (label <- labels) yield bias.Factor(label) ++
    for (label <- labels) yield obs.Factor(label, label.token) ++
    for (label <- labels.drop(1)) yield markov.Factor(label.prev, label)
  }
// Learning
val trainer = new HogwildTrainer(AROW, model)
trainer.process(sentences.map(y => new SampleRankPiece(y))

// Inference
val marginals = BP.infer(sentences.first, model)
Example: Sequence Labeling
“hogwild”..., with Markov deps, Sparse BP inference

// Data
class Token extends FeatureVectorVariable { def domain = DocumentDomain }  
class Label(val doc:Document) extends LabeledCategoricalVariable { def 
val sentences: Iterable[Seq[Label]] = readData()  
// Model
var model = new Model[Seq[Label]] {  
  val bias = new DotFamilyWithStatistics1[Label] { val weights = new 
  val obs = new DotFamilyWithStatistics[Label,Document] { val weights = 
  val markov = new DotFamilyWithStatistics2[Label,Label] { val weights 
    def factors(labels:Seq[Label]) = 
      for (label <- labels) yield bias.Factor(label) ++ 
      for (label <- labels) yield obs.Factor(label, label.token) ++ 
      for (label <- labels.drop(1)) yield markov.Factor(label.prev, label 
  }  
// Learning
val trainer = new HogwildTrainer(AROW, model)  
trainer.process(sentences.map(y => new SampleRankPiece(y))  
// Inference
val marginals = SparseBP.infer(sentences.first, model)
FACTORIE Progress This Year

• Design
  - *Model* now source of factors with arbitrary context
  - *Factor* no long tied to Family
  - *Tensors* efficient multi-dimensional linear algebra
  - *Cubbies* for serialization to files and No-SQL DBs

• New application frameworks:
  - NLP representations and pre-trained models
  - Linear chains
  - Classification
  - Topic models

• New learning & joint inference algorithms
  - Parameter estimation flexibility, based on “Pieces”.
  - New inference procedures, e.g. Sparse BP.

• Released version 1.0, milestone 1.
  - Growing tutorial documentation.
FACTORIE Current Usage

• Boyan Onyshkevych (NSA) in collaboration with John Frank (MIT) and collaborators at NIST to support NIST TAC-KBA and internal work on large-scale entity resolution.

• DoD work at BAE, SRI, SAIC, Digital Reasoning Systems Inc., Oculus Inc., and Decisive Analytics Corporation,...

• Corporate work at Oracle, Google, Contata,...

• Academic work at UMass, Stanford, CMU, UMD,...
Outline

• **New algorithms for joint inference**
  - Guaranteed-optimal “beam search” by Column Generation [NIPS 2012]
  - Joint inference with sparse values & factors [Submitted 2013]
  - Compressing messages with Expectation Propagation [TR 2012]
  - Focussed Query-specific MCMC Sampling [NIPS 2011]
  - Joint reasoning about relations of “universal schema” [AKBC 2012]

• **FACTORIE** Probabilistic programming support for joint inference
  - Landscape of tools
  - Capabilities
  - Release 1.0