Motivation / Project Goal

Project goal: Build a trainable dependency parser that is easily portable to many languages (given annotated training data).

Application: Microsoft Research's Machine Translation System:

Outline

1. Intro to Dependency Parsing
2. Dependency Parsing as “Structured Classification”
3. Parser architecture
4. Training by Bayes Point Machines
5. Experiments

Constituency vs. Dependency Parsing

Constituency parse:
- indicates phrase structures
- context free grammar rules

Dependency parse:
- relationships between words
- arrow indicates head-child relations
- e.g. "hot" modifies "peppers"
- e.g. "peppers" is argument of "like"

Why Dependency Parsing?

- Some NLP systems need only word-to-word relationship information, e.g.:
  - Machine translation [Quirk et.al., ACL 05]
  - Information extraction [Bunescu&Mooney, HLT05]
  - Question answering [Punyakanok et.al, AIMath04]
- Ease of annotation
  - No grammar building
  - Native speakers can do the job

Dependency Parsing for different languages

- Projective dependency parses
- Non-projective: (crossing arrows)

- Free word-order languages (e.g. Czech, Arabic) have more non-projective trees
  - Czech treebank: 25% sentences, 2% dependencies, (Nivre, 2005)

Outline

1. Intro to Dependency Parsing
2. Dependency Parsing as “Structured Classification”
3. Parser architecture
4. Training by Bayes Point Machines
5. Experiments

Some NLP systems need only word-to-word relationship information, e.g.: Machine translation, Information extraction, Question answering, Ease of annotation.
Outline

1. Intro to Dependency Parsing
2. Dependency Parsing as “Structured Classification”
3. Parser architecture
4. Training by Bayes Point Machines
5. Experiments

Challenges

\[ \hat{y} = \text{arg max } F(x, y) \]

- How to define and learn \( F(x, y) \)?
- How to efficiently compute \( \text{ArgMax} \)?

“Structured classification”

- Conventional classification problem:
  \[ x \xrightarrow{F()} y \]
  \( x \) : vector of input features
  \( y \) : scalar output

- Structured classification problem:
  \[ x \xrightarrow{F()} y \]
  \( x \) : vector of input features
  \( y \) : complex set of outputs (e.g. vector, parse)
  values in output may be interdependent

- Popular solutions:
  - Graphical models
  - M3 Nets (Taskar), Structured SVM (Joachims)

Dependency Parsing as Structured Classification

- Input: features of a sentence
- Output: a whole dependency parse

- Structure constraints: parse is a directed acyclic graph (tree) spanning all words

A Solution to Structured Classification

\[ \hat{y} = \text{arg max } F(x, y) \]

\( y \in \text{GEN}(x) \)

- \( x \) : input sentence
- \( \text{GEN}(x) \) : generates all possible parses of \( x \)
- \( F(x, y) \) : function that scores a parse
- \( \text{ArgMax} \) : choose output with the best parse

Outline

1. Intro to Dependency Parsing
2. Dependency Parsing as “Structured Classification”
3. Parser architecture
   1. Defining \( F(x, y) \)
   2. ArgMax implementation (Decoder)
4. Training by Bayes Point Machines
5. Experiments
Defining $F(x,y)$: decomposition

$$\arg \max_{y \in \text{GEN}(x)} F(x,y) \rightarrow \arg \max_{y \in \text{GEN}(x)} \sum_{(i,j) \in y} \text{score}(i,j)$$

Input: sentence and scores of edges
Output: parse with max $F(x,y)$

Parser Architecture: 3 components

$$\arg \max_{y \in \text{GEN}(x)} \sum_{(i,j) \in y} w \cdot h(i,j)$$

Decoder/ARGMAX

- Requirements:
  - Must search all possible parses for a given sentence
  - Must search fast
    - ArgMax will be invoked multiple times in discriminative training
    - (Preferably) Don’t do exhaustive search, don’t enumerate malformed parse
- We used:
  - Eisner’s decoder for projective trees [Eisner, ACL96]
  - Chu-Liu-Edmonds decoder for non-projective [McDonald, et.al. HLT2005]

Outline

1. Intro to Dependency Parsing
2. Dependency Parsing as “Structured Classification”
3. Parser architecture
4. Training by Bayes Point Machines
   1. Version Space
   2. BPM: Bayesian averaging of classifiers
5. Experiments

Defining $F(x,y)$: edge scores

$$\arg \max_{y \in \text{GEN}(x)} \sum_{(i,j) \in y} \text{score}(i,j) \rightarrow \arg \max_{y \in \text{GEN}(x)} \sum_{(i,j) \in y} w \cdot h(i,j)$$

$h(i,j)$: feature vector of pair word $i$ and word $j$
- define based on linguistic knowledge
- specify different features for different languages
- $w$: weights
  - trained by machine learning methods (discriminatively)

Weight space/Feature space

Duality and Version Space

$x$: feature space
$w$: weight space

Point in weight space $$\leftrightarrow$$ hyperplane in feature space
Point in feature space $$\leftrightarrow$$ hyperplane in weight space (defines a halfspace)
Bayes Point Machines (Herbrich, 2001)

- **Motivation:**
  - Bayesian averaging of classifiers
  - Find the Center-of-Mass solution ($W_{cm}$)

- **Main Idea:**
  1. Approximate $W_{cm}$ by sampling the version space
  2. Sampling is achieved by running perceptron training on randomly shuffled data
  3. Each perceptron gives a $w$, which is then combined to form the BPM solution

### BPM Equations

- **Ideal Bayesian averaging to achieve $W_{cm}$:**
  \[
  w_{BPM} = E_{p(w|D)}[w] = \sum_k p(w_k|D)w_k
  \]

- **In practice…**
  
  - version space is large => take finite sample of $w$
  
  - assume uniform prior $p(w)$

  \[
  w_{BPM} = E_{p(w|D)}[w] = \frac{1}{K} \sum_{k=1}^{K} w_k
  \]

### BPM Pseudo-code

- **INPUT:**
  - $x_i$: set of training points, $i=1, ..., N$
  - $y_i \in \{-1, 1\}$ : labels of $x_i$

- **OUTPUT:**
  - $w$: discriminatively trained weight vector
  - Linear model: $\hat{y}_i = \text{sign}(w \cdot x_i)$

\[
\begin{align*}
0. & \quad \text{for } k = 1:K \\
1. & \quad \text{Initialize } w_k=0; \text{ Randomly shuffle training data} \\
2. & \quad \text{for } i = 1: N \\
3. & \quad \quad \hat{y}_i = \text{sign}(w_k \cdot x_i) \\
4. & \quad \quad \text{if } \hat{y}_i \neq y_i \\
5. & \quad \quad \quad w_k = w_k + y_i x_i \\
6. & \quad \text{Repeat until convergence} \\
7. & \quad \text{end} \\
8. & \quad w = \frac{1}{K} \sum_{k=1}^{K} w_k
\end{align*}
\]

### Bayes Point Machine

#### Pros & Cons

- **Pros:**
  
  - Good generalization
  
  - Online learning
  
  - Easy to implement
  
  - Parallel computation

- **Cons:**
  
  - Sampling scheme is only approximate
  
  - Computation grows with number of perceptrons

#### Outline

1. Intro to Dependency Parsing
2. Dependency Parsing as “Structured Classification”
3. Parser architecture
4. Discriminative Training of Parameters
5. Experiments
   1. Data & Features
   2. Evaluation on English, Czech, Arabic, Chinese
### Data

<table>
<thead>
<tr>
<th>Language</th>
<th>Tokens</th>
<th>Train Sent</th>
<th>Test Sent</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arabic</td>
<td>116k</td>
<td>2100</td>
<td>449</td>
<td>Prague Arabic Dependency Treebank (v1)</td>
</tr>
<tr>
<td>Chinese</td>
<td>527k</td>
<td>14k</td>
<td>2080</td>
<td>Chinese Treebank (v5)</td>
</tr>
<tr>
<td>Czech</td>
<td>1.6M</td>
<td>73k</td>
<td>7507</td>
<td>Prague Dependency Treebank (v1)</td>
</tr>
<tr>
<td>English</td>
<td>1M</td>
<td>40k</td>
<td>2416</td>
<td>Penn Treebank</td>
</tr>
</tbody>
</table>

### Evaluation

**Evaluation Measures:**
- Dependency Accuracy
- Root Accuracy/F1
- Complete Accuracy

*Report dependency acc with/without punctuation*

*What's best depends on application, e.g.:*
- If used for semantic analysis, no need for punctuation
- If used for sentence simplification, need punctuation

### BPM vs. Perceptrons

<table>
<thead>
<tr>
<th></th>
<th>Arabic</th>
<th>Chinese</th>
<th>Czech</th>
<th>English</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bayes Point Machine</td>
<td>78.4</td>
<td>83.8</td>
<td>-</td>
<td>91.2</td>
</tr>
<tr>
<td>Best Perceptron</td>
<td>77.9</td>
<td>83.1</td>
<td>83.8</td>
<td>90.8</td>
</tr>
<tr>
<td>Worst Perceptron</td>
<td>77.4</td>
<td>82.6</td>
<td>83.7</td>
<td>90.5</td>
</tr>
</tbody>
</table>

**Observation:**
BPM result is always better than the best perceptron
=> averaging classifiers works!

### Features

- Extract for every given pair of dependencies in Training Set:
  - ParentToken
  - ChildToken
  - ParentPOS
  - ChildPOS
  - POS of intervening words
- Backoff features:
  - Czech/English: first five characters “stem”
  - Arabic: stem from a morphological analyzer
  - Chinese: first character “stem”
- Combinations of above to achieve “polynomial kernels”

### Comparison to state-of-the-art

**BPM better than MIRA in Complete Acc, worse in Dependency/Root Acc.**

<table>
<thead>
<tr>
<th></th>
<th>English</th>
<th>Dependency Accuracy</th>
<th>Root Accuracy</th>
<th>Complete Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perceptron</td>
<td>90.6</td>
<td>94.0</td>
<td>36.5</td>
<td></td>
</tr>
<tr>
<td>MIRA (McDonald, 05)</td>
<td><strong>90.9</strong></td>
<td><strong>94.2</strong></td>
<td><strong>37.5</strong></td>
<td></td>
</tr>
<tr>
<td>Bayes Point Machines</td>
<td>90.8</td>
<td>93.7</td>
<td><strong>37.6</strong></td>
<td></td>
</tr>
</tbody>
</table>

### Comparing results across languages

#### With Punctuation

<table>
<thead>
<tr>
<th>Language</th>
<th>DA</th>
<th>RA</th>
<th>CM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arabic</td>
<td>79.9</td>
<td>90.0</td>
<td>9.80</td>
</tr>
<tr>
<td>Chinese</td>
<td>71.2</td>
<td>66.2</td>
<td>17.5</td>
</tr>
<tr>
<td>Czech</td>
<td>83.3</td>
<td>88.3</td>
<td>29.2</td>
</tr>
<tr>
<td>English</td>
<td>90.0</td>
<td>93.7</td>
<td>35.1</td>
</tr>
</tbody>
</table>

#### Without Punctuation

<table>
<thead>
<tr>
<th>Language</th>
<th>DA</th>
<th>RA</th>
<th>CM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arabic</td>
<td>79.8</td>
<td>87.8</td>
<td>10.2</td>
</tr>
<tr>
<td>Chinese</td>
<td>73.3</td>
<td>66.2</td>
<td>18.2</td>
</tr>
<tr>
<td>Czech</td>
<td>83.6</td>
<td>75.5</td>
<td>30.1</td>
</tr>
<tr>
<td>English</td>
<td>90.8</td>
<td>93.7</td>
<td>37.6</td>
</tr>
</tbody>
</table>

**What makes accuracy vary for different languages?**
- language characteristics (e.g. inflectional morphology leading to data sparsity)
- annotation scheme
- training data size
Comparing results across languages: Data reduction exp.

Observations:
- At all sample sizes, English wins
- Czech has worse results, but improves with more data (not shown)
- Why does Chinese and Arabic have similar results?

Summary/Conclusions

- View Dependency Parsing as “Structured Classification”
  \[ \arg \max_{y \in \mathcal{Y}(x)} F(x, y) \rightarrow \arg \max_{y \in \mathcal{Y}(x)} \sum_{(i,j)} w \cdot \delta(i,j) \]
- Bayes Point Machine training
  - Bayesian averaging of classifiers => \( W_m \)
  - As simple to implement as the perceptron, yet competitive with large margin methods
- Results in four different languages
  - Further work on cross-language comparison needed

Thank you!

- Questions?

Data (more)

English:
- Penn Treebank
- Extract dependencies by Yamada&Matsumoto (IWPT03) heuristics
- POS: use human-annotation for training, Toutanova’s tagger for test

Chinese:
- Chinese treebank (v5)
- Extract dependencies using heuristics
- POS: use human-annotation for training, Toutanova’s tagger for test
  (tagger has 92.0% token accuracy, 63.8% sentence accuracy on devset)

Czech:
- Prague Dependency Treebank (v1)
- use human-annotated POS & auto-tagged morphological info in train/test

Arabic:
- Prague Arabic Dependency Treebank (v1)
- use human-annotated POS & auto-tagged morphological info in train/test