NLP Structures

- Trees

```
S
   NP  VP
  |    |
 John saw NP
     |    |
      Mary
```

- Tagged sequences, e.g., named entity tagging

```
S — C — N — N — N — S
 Napoleon Bonaparte was exiled to Elba
```

S = Start entity
C = Continue entity
N = Not an entity
Feature Vectors: $\Phi$

- $\Phi$ defines the **representation** of a structure
- $\Phi$ maps a structure to a **feature vector** $\in \mathbb{R}^d$

\[ \downarrow \Phi \]

\[ \langle 1, 0, 2, 0, 0, 15, 5 \rangle \]
Features

- A “feature” is a function on a structure, e.g.,

\[ h(x) = \text{Number of times} \quad \begin{array}{c}
\text{A} \\
\text{B} \\
\text{C}
\end{array} \quad \text{is seen in } x \]

\[ T_1 \quad \begin{array}{c}
\text{A} \\
\text{B} \\
\text{C} \\
\text{D} \\
\text{E} \\
\text{F} \\
\text{G}
\end{array} \quad \begin{array}{c}
\text{d} \\
\text{e} \\
\text{f} \\
\text{g}
\end{array} \]

\[ T_2 \quad \begin{array}{c}
\text{A} \\
\text{B} \\
\text{C} \\
\text{D} \\
\text{E} \\
\text{F} \\
\text{A}
\end{array} \quad \begin{array}{c}
\text{d} \\
\text{e} \\
\text{h} \\
\text{b} \\
\text{c}
\end{array} \]

\[ h(T_1) = 1 \quad h(T_2) = 2 \]
Feature Vectors

- A set of functions $h_1 \ldots h_d$ define a feature vector

$$\Phi(x) = \langle h_1(x), h_2(x) \ldots h_d(x) \rangle$$

\[ \begin{align*}
\Phi(T_1) &= \langle 1, 0, 0, 3 \rangle \\
\Phi(T_2) &= \langle 2, 0, 1, 1 \rangle
\end{align*} \]
“All Subtrees” Representation [Bod, 1998]

- Given: Non-Terminal symbols \{A, B, \ldots\}
  Terminal symbols \{a, b, c \ldots\}

- An infinite set of subtrees

  \[
  \begin{array}{c}
  A \\
  B \\
  b
  \end{array}
  \quad
  \begin{array}{c}
  A \\
  B \\
  E \\
  b
  \end{array}
  \quad
  \begin{array}{c}
  A \\
  B \\
  C \\
  A \\
  B \\
  b
  \end{array}
  \quad
  \begin{array}{c}
  A \\
  B \\
  A
  \end{array}
  \quad
  \ldots
  \]

- An infinite set of features, e.g.,

  \[h_3(x, y) = \text{Number of times } \quad \begin{array}{c}
  A \\
  B \\
  b
  \end{array} \quad \text{is seen in } (x, y)\]
All Sub-fragments for Tagged Sequences

- Given: State symbols $\{S, C, N\}$
  Terminal symbols $\{a, b, c, \ldots\}$

- An infinite set of sub-fragments

  $S \quad S \quad S \quad \text{—} \quad C \quad S \quad \text{—} \quad C$
  
  \[ \begin{array}{c}
  S \quad S \\
  a \quad b \\
  \end{array} \quad \ldots \]

- An infinite set of features, e.g.,

  $h_3(x) = \text{Number of times} \begin{array}{c}
  S \quad \text{—} \quad C \\
  b \\
  \end{array} \text{ is seen in } x$
Inner Products

- \( \Phi(x) = \langle h_1(x), h_2(x) \ldots h_d(x) \rangle \)

- Inner product ("Kernel") between two structures \( T_1 \) and \( T_2 \):
  \[
  \Phi(T_1) \cdot \Phi(T_2) = \sum_{i=1}^{d} h_i(T_1) h_i(T_2)
  \]

\[
\begin{align*}
T_1 & \quad A \\
B & \quad C \\
D & \quad E & F &\quad G \\
d & \quad e & f & \quad g \\
\end{align*}
\]

\[
\begin{align*}
T_2 & \quad A \\
B & \quad C \\
D & \quad E & F &\quad A \\
d & \quad e & h & \quad B & C \\
& \quad b & c
\end{align*}
\]

\[
\Phi(T_1) = \langle 1, 0, 0, 3 \rangle \\
\Phi(T_2) = \langle 2, 0, 1, 1 \rangle \\
\Phi(T_1) \cdot \Phi(T_2) = 1 \times 2 + 0 \times 0 + 0 \times 1 + 3 \times 1 = 5
\]
“All Subtrees” Representation

- Given: Non-Terminal symbols \( \{A, B, \ldots\} \)
  Terminal symbols \( \{a, b, c \ldots\} \)

- An infinite set of subtrees

\[
\begin{align*}
&\text{Step 1:} \\
&\quad \text{Choose an (arbitrary) mapping from subtrees to integers} \\
&\quad h_i(x) = \text{Number of times subtree } i \text{ is seen in } x \\
&\quad \Phi(x) = \langle h_1(x), h_2(x), h_3(x) \ldots \rangle
\end{align*}
\]
All Subtrees Representation

- $\Phi$ is now huge

- But inner product $\Phi(T_1) \cdot \Phi(T_2)$ can be computed efficiently using dynamic programming.
Computing the Inner Product

Define $N_1$ and $N_2$ are sets of nodes in $T_1$ and $T_2$ respectively.

$$ - I_i(x) = \begin{cases} 1 \text{ if } i\text{'th subtree is rooted at } x. \\ 0 \text{ otherwise.} \end{cases} $$

Follows that:

$$ h_i(T_1) = \sum_{n_1 \in N_1} I_i(n_1) \quad \text{and} \quad h_i(T_2) = \sum_{n_2 \in N_2} I_i(n_2) $$

$$ \Phi(T_1) \cdot \Phi(T_2) = \sum_i h_i(T_1)h_i(T_2) = \sum_i \left( \sum_{n_1 \in N_1} I_i(n_1) \right) \left( \sum_{n_2 \in N_2} I_i(n_2) \right) $$

$$ = \sum_{n_1 \in N_1} \sum_{n_2 \in N_2} \sum_i I_i(n_1)I_i(n_2) $$

$$ = \sum_{n_1 \in N_1} \sum_{n_2 \in N_2} \Delta(n_1, n_2) $$

where $\Delta(n_1, n_2) = \sum_i I_i(n_1)I_i(n_2)$ is the number of common subtrees at $n_1, n_2$.
An Example

\[ \Phi(T_1) \cdot \Phi(T_2) = \Delta(A, A) + \Delta(A, B) \ldots + \Delta(B, A) + \Delta(B, B) \ldots + \Delta(G, G) \]

- Most of these terms are 0 (e.g. \( \Delta(A, B) \)).
- Some are non-zero, e.g. \( \Delta(B, B) = 4 \)
Recursive Definition of $\Delta(n_1, n_2)$

- If the productions at $n_1$ and $n_2$ are different
  \[ \Delta(n_1, n_2) = 0 \]

- Else if $n_1, n_2$ are pre-terminals,
  \[ \Delta(n_1, n_2) = 1 \]

- Else
  \[ \Delta(n_1, n_2) = \prod_{j=1}^{nc(n_1)} (1 + \Delta(ch(n_1, j), ch(n_2, j))) \]
  
  $nc(n_1)$ is number of children of node $n_1$;
  $ch(n_1, j)$ is the $j$’th child of $n_1$. 
Illustration of the Recursion

How many subtrees do nodes $A$ and $A$ have in common? i.e., What is $\Delta(A, A)$?

$\Delta(B, B) = 4$

$\Delta(C, C) = 1$

$\Delta(A, A) = (\Delta(B, B) + 1) \times (\Delta(C, C) + 1) = 10$
The Inner Product for Tagged Sequences

- Define $N_1$ and $N_2$ to be sets of states in $T_1$ and $T_2$ respectively.

- By a similar argument,

$$\Phi(T_1) \cdot \Phi(T_2) = \sum_{n_1 \in N_1} \sum_{n_2 \in N_2} \Delta(n_1, n_2)$$

where $\Delta(n_1, n_2)$ is number of common sub-fragments at $n_1, n_2$

\[\begin{align*}
\text{e.g., } T_1 &= \begin{array}{cccc}
A & B & C & D \\
a & b & c & d
\end{array} &
T_2 &= \begin{array}{cccc}
A & B & C & E \\
a & b & b & e & e
\end{array} \\
\Phi(T_1) \cdot \Phi(T_2) &= \Delta(A, A) + \Delta(A, B) + \Delta(B, A) + \Delta(B, B) + \Delta(D, E) \\
\text{e.g., } \Delta(B, B) &= 4,
\end{align*}\]
The Recursive Definition for Tagged Sequences

- Define $N(n) =$ state following $n$, $W(n) =$ word at state $n$
- Define $\pi[W(n_1), W(n_2)] = 1$ iff $W(n_1) = W(n_2)$
- Then if labels at $n_1$ and $n_2$ are the same,

$$\Delta(n_1, n_2) = (1 + \pi[W(n_1), W(n_2)]) \times (1 + \Delta(N(n_1), N(n_2)))$$

e.g., $T_1 =$

```
A — B — C — D
a   b   c   d
```

$T_2 =$

```
A — B — C — E
a   b   e   e
```

$$\Delta(A, A) = (1 + \pi[a, a]) \times (1 + \Delta(B, B))$$

$$= (1 + 1) \times (1 + 4) = 10$$
Refinements of the Kernels

- Include log probability from the baseline model:

  \( \Phi(T_1) \) is representation under all sub-fragments kernel

  \( L(T_1) \) is log probability under baseline model

New representation \( \Phi' \) where

\[
\Phi'(T_1) \cdot \Phi'(T_2) = \beta L(T_1)L(T_2) + \Phi(T_1) \cdot \Phi(T_2)
\]

(includes \( L(T_1) \) as an additional component with weight \( \sqrt{\beta} \))

- Allows the perceptron to use original ranking as default
Refinements of the Kernels

- Downweighting larger sub-fragments

\[ \sum_{i=1}^{d} \lambda^{SIZE_i} h_i(T_1)h_i(T_2) \]

where \(0 < \lambda \leq 1\),
\(SIZE_i\) is number of states/rules in \(i\)'th fragment

- Simple modification to recursive definitions, e.g.,

\[ \Delta(n_1, n_2) = (1 + \pi[W(n_1), W(n_2)]) \times (1 + \lambda \Delta(N(n_1), N(n_2))) \]
Refinement of the Tagging Kernel

- Sub-fragments sensitive to spelling features (e.g., Capitalization)

- Define $\pi[x, y] = 1$ if $x$ and $y$ are identical, $\pi[x, y] = 0.5$ if $x$ and $y$ share same capitalization features

$$\Delta(n_1, n_2) = (1 + \pi[W(n_1), W(n_2)]) \times (1 + \lambda \Delta(N(n_1), N(n_2)))$$

- Sub-fragments now include capitalization features

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>exiled</td>
<td>to Elba</td>
<td>exiled</td>
</tr>
<tr>
<td>No cap</td>
<td>to Cap</td>
<td>No cap</td>
</tr>
</tbody>
</table>
## Experimental Results

### Parsing Wall Street Journal

<table>
<thead>
<tr>
<th>MODEL</th>
<th>( \leq 100 \text{ Words (2416 sentences)} )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LR</td>
</tr>
<tr>
<td>CO99</td>
<td>88.1%</td>
</tr>
<tr>
<td>VP</td>
<td>88.6%</td>
</tr>
</tbody>
</table>

VP gives 5.1% relative reduction in error (CO99 = my thesis parser)

### Named Entity Tagging on Web Data

<table>
<thead>
<tr>
<th></th>
<th>P</th>
<th>R</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max-Ent</td>
<td>84.4%</td>
<td>86.3%</td>
<td>85.3%</td>
</tr>
<tr>
<td>Perc.</td>
<td>86.1%</td>
<td>89.1%</td>
<td>87.6%</td>
</tr>
<tr>
<td>Improvement</td>
<td>10.9%</td>
<td>20.4%</td>
<td>15.6%</td>
</tr>
</tbody>
</table>

VP gives 15.6% relative reduction in error
Summary

• For any representation $\Phi(x)$,
  Efficient computation of $\Phi(x) \cdot \Phi(y) \Rightarrow$
  Efficient learning through kernel form of the perceptron

• Dynamic programming can be used to calculate $\Phi(x) \cdot \Phi(y)$
  under “all sub-fragments” representations

• Several refinements of the inner products:
  – Including probabilities from baseline model
  – Downweighting larger sub-fragments
  – Sensitivity to spelling features