Natural Language Processing

Discriminative models for syntactic parsing
<table>
<thead>
<tr>
<th>Method</th>
<th>Tagging</th>
<th>Parsing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generative</td>
<td>HMMs</td>
<td>PCFGs</td>
</tr>
<tr>
<td>Log-linear</td>
<td>Greedy</td>
<td>Transition-based parsing</td>
</tr>
<tr>
<td></td>
<td>History-based</td>
<td>Viterbi</td>
</tr>
<tr>
<td></td>
<td>Global</td>
<td>forward-backward</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CKY</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Inside-outside</td>
</tr>
</tbody>
</table>
Motivation

• Incorporate arbitrary overlapping features for discrimination

• Tagging:
  • word/tag pairs
  • prefixes <5/tag pairs
  • suffixes <5/tag pairs
  • tag trigrams
  • tag bigrams
  • tag unigrams
  • word/tag pairs for previous word
  • word/tag pairs for subsequent word
Motivation

• Parsing (CRF parsing in this case):
  • Conjunction of rule and
    • first word, last word, word before split, word after split, word before, word after
  • Replace rule with parent category
  • Span shape
  • Span length (binned)
Log-linear models

• **Input:** domain $X$, label set $Y$

• **Feature function:**
  \[ f : X \times Y \rightarrow \mathbb{R}^m \]

• **Parameters:**
  \[ \theta \in \mathbb{R}^m \]

• **Output:** $p(y \mid x)$
  \[
  p_\theta(y \mid x) = \frac{e^{f(x,y)\top \theta}}{\sum_{y' \in Y} e^{f(x,y')\top \theta}}
  \]
Plan

• Locally normalized models
  • AKA, transition-based, greedy parsers
• Globally normalized models (CRF)
• Neural CRF
Locally-normalized
Log-linear models for tagging

- Input: \( w_1 \ldots w_n \)
- Output: \( t_1 \ldots t_n \)

\[
p(t_1, \ldots, t_n \mid w_1, \ldots, w_n, t_{-1}, t_0) = \prod_{i=1}^{n} p(t_i \mid t_{-1}, \ldots, t_{i-1}, w_1, \ldots, w_n)
\]

- **Learning**: estimate \( p(t_i \mid t_{i-2}, t_{i-1}, w_1, \ldots w_n) \)
- **Decoding**: Viterbi
  - \( \text{argmax}_{t_1 \ldots t_n} \log p(t_1 \ldots t_n \mid w_1 \ldots w_n, t_{-1}, t_0) \)
Log-linear models for parsing

- Goal: define a model for $p(t \mid s)$
- $t$ is a tree
- $s$ is the sentence $w_1 \ldots w_n$
- **Difficulty:** In tagging, we made tagging decisions left-to-right. What is the order for parsing?
Outline of approach

- Represent a tree as a sequence of $m$ decisions:
  \[ T = (d_1, d_2, \ldots, d_m) \]

- Tree probability:
  \[ p(t \mid s) = \prod_{i=1}^{m} p(d_i \mid d_1, \ldots, d_{i-1}, s) \]

- Decision probability is a log-linear model:
  \[ p(d_i \mid d_1, \ldots, d_{i-1}, s) = \frac{e^{f(d_1, \ldots, d_{i-1}, d_i, s) \top \theta}}{\sum_{d'} e^{f(d_1, \ldots, d_{i-1}, d', s) \top \theta}} \]

- Search for high(est?) probability tree based on $p(t \mid s)$
Decomposing a tree
Ratnaparkhi, 1997

- Bottom-up, left-to-right transition system
- Three layers of decisions
  - Part-of-speech tags
  - Chunks
  - The rest
Search

- In POS tagging we could use Viterbi because we used a Markov assumption

\[
p(t_1, \ldots, t_n \mid w_1, \ldots, w_n, t_{-1}, t_0) = \prod_{i=1}^{n} p(t_i \mid t_{-1}, \ldots, t_{i-1}, w_1, \ldots, w_n)
\]

- Here features depend on arbitrary past decisions, and thus dynamic programming is not used
Beam search

- Keep K possible decision sequences at every point
  - Expand the K sequences in all possible ways
  - Keep the top-K ones for the next step
- Stop once you have M completed parse trees
Evaluation

- Slightly worse than lexicalized PCFGs (86.9 F₁)
- But is faster! (linear empirically)
Shift-reduce parser

- Another greedy transition-based parser

- Linear time complexity

- Based on a transition-based parser for dependency parsing, very popular and useful nowadays

- Supports unary and binary rules

- The main difference is the transition system
Data structures

- **Buffer $B$:** holds pos-tagged words to be processed
- **Stack $S$:** hold partially constructed trees
- *the man ate an apple*
Initialization and termination

- **Initialization**: entire sentence is in the buffer (first word at the front, empty stack)

- **Termination**: buffer is empty, stack has one tree, which is the output

```
DT     NN    VBD   DT    NN
the    man    ate    an   apple
```

```
B

S
```

```
NP(man)

VP(ate)

NP(apple)

DT(the)  NN(man)  VBD(ate)  DT(an)  NN(apple)

the    man    ate    DT(an)  NN(apple)
```
Actions: shift

- **Shift**: remove pos-tagged word from buffer and push to the stack

\[\text{ate} \quad \text{an} \quad \text{apple} \]

\[\text{NP(man)} \quad \text{DT(the)} \quad \text{NN(man)}\]
Actions: reduce-X-left

- pop from the stack the top two trees. Construct new tree with category X, with the top tree as right child and bottom tree as left child. Copy head from the left child.
Actions: reduce-X-right

- pop from the stack the top two trees. Construct new tree with category X, with the top tree as right child and bottom tree as left child. Copy head from the right child.
Actions: reduce-X-unary

- pop from the stack a tree. Construct new tree with category X with the popped tree as a child. Copy the head.
# Action sequence

- **The men ate an apple**

<table>
<thead>
<tr>
<th>Buffer</th>
<th>Stack</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>The men ate an apple</td>
<td>the</td>
<td>shift</td>
</tr>
<tr>
<td>men ate an apple</td>
<td>men</td>
<td>the</td>
</tr>
<tr>
<td>ate an apple</td>
<td>(NP the men)</td>
<td>shift</td>
</tr>
<tr>
<td>an apple</td>
<td>ate</td>
<td>(NP the men)</td>
</tr>
<tr>
<td>apple</td>
<td>an</td>
<td>ate</td>
</tr>
<tr>
<td></td>
<td>apple</td>
<td>an</td>
</tr>
<tr>
<td></td>
<td>(NP an apple)</td>
<td>ate</td>
</tr>
<tr>
<td></td>
<td>(VP ate an apple)</td>
<td>(NP the men)</td>
</tr>
<tr>
<td></td>
<td>(S the men ate an apple)</td>
<td>Reduce-S-right</td>
</tr>
</tbody>
</table>
Justification/Proof

• Any parse tree can be constructed with a sequence of these actions (let’s ignore unaries)

• **Height of a tree**: number of edges on longest path from root to leaves (POS tags)
  
  • **Height 0**: single word sentences. Obtained by a single shift action
  
  • **Height 1**: only one binary tree is possible (adding unaries is easy) and is obtained by the action sequence of shift-shift-reduce

• Assume we can build tree of length < \( n \). **For height \( n \):**
  
  • Construct the left child. Now there is one item in the stack
  
  • Construct the right child. Now there are two items in the stack and an empty buffer.

  • reduce
Training

• As usual:
  • Input: \( s, d_1, \ldots, d_{i-1} \)
  • Output: \( d_i \)

• Learning:
  • Sagae and Lavie use the perceptron learning algorithm
  • Log-linear models
  • Neural network
Features

• Based on
  • Top four items on stack
  • Top four items on buffer
  • Standard binary features (non-terminals at root and children, POS tags, words, children, …)
• What if you want to use a neural network?
Complexity

• For a sentence with $n$ words
  • $n$ shift actions
  • $n-1$ binary reductions
  • Do not allow more than $k (=3)$ consecutive unary reduction actions
• Linear number of actions
Evaluation

- 87.5 F1 (slightly lower than best LPCFG at the time)
- ~3-4x faster than a lexicalized PCFG parser
Globally-normalized
Globally-normalized models

- Given a training set \((x, y)\) of pairs of sentences with their parse trees

\[
p_\theta(y \mid x) = \frac{\exp(f(x, y) \top \theta)}{\sum_{y' \in \mathcal{Y}} \exp(f(x, y') \top \theta)}
\]

\(\mathcal{Y} : \) all parse trees for \(x\)
Global linear models

- Motivations:
  - Optimize what you care about
  - **Flexibility in feature function**
  - Avoid the label bias problem
Globally-normalized models

- Contain three components
  - A feature function $f$ mapping a pair $(x, y)$ to a feature vector $f(x, y)$
  - A generating function $GEN(x)$ enumerating all candidate outputs
  - A parameter vector $\theta$

$$F(x) = \arg \max_y f(x, y)^\top \theta$$
Parse re-ranking

• Assume we define GEN(x) be a set of top-K parse trees given by a lexicalized PCFG

• There is no decoding! This is a re-reranking

• We can define any feature over the parse tree
Example: coordination

- Parallelism in coordination
  - *Bars in New-York and pubs in London*
  - *Bars in New-York and pubs*

- It is not trivial to featurize this with a PCFG (even lexicalized). Easy to do given the full parse tree
  - Count the number of parallel/non-parallel coordination structures
Example: complex VPs

• No need to binarize, since there is no decoding
  • For a rule VP—> PP VBD NP NP SBAR
    • All adjacent non-terminal bigrams to the left and right of the head
      • <s> PP, NP NP, NP SBAR, SBAR </s>
    • Grandparent features
      • S—>VP—> PP VBD NP NP SBAR
    • Sibling features
      • S—>VP—> PP VBD NP NP SBAR and
      • S—>NP
What if want to parse from scratch?

We need to use CKY-style algorithms for learning and decoding

Features need to decompose
Rule productions

• Assume an input sentence $x$

• Assume a CFG in CNF $G=(V, \Sigma, R, S)$

• The CFG defines a set of parse trees $Y$

• A tree $y$ in $Y$ is a set of rule productions of the form:
  
  • $(A, B, C, i, k, j)$: the rule $A \rightarrow BC$ is seen with non-terminal $A$ spanning $x_i \ldots x_j$, non-terminal $B$ spanning $x_i \ldots x_k$, and non-terminal $C$ spanning $x_{k+1} \ldots x_j$
  
  • $(A, i)$: The rule $A \rightarrow x_i$ is in the parse tree
Example

S, NP VP, 1, 2, 5
NP, NN, NNS, 1, 1, 2
VP, VB, NP, 3, 3, 5
NP, DT, NN, 4, 4, 5
NN, 1
NNS, 2
VB, 3
DT, 4
NN, 5
Class 9
Warning: all feature functions can look at $x$, I didn’t always consistently write that.

Features decompose over rule productions:

$$ f(x, y) = \sum_{r \in y} g(r)^\top \theta = \sum_{r \in y} g(x, A, B, C, i, j, k)^\top \theta $$

Example features:

- The PCFG rule used
- Conjunction of LHS with $x_i$, $x_{i-1}$, $x_j$, $x_{j+1}$, …
Global linear model

\[ p_\theta(y \mid x) = \frac{\exp(f(x, y)\top \theta)}{\sum_{y' \in \text{GEN}(x)} \exp(f(x, y')\top \theta)} \]

\[ = \frac{\exp(\sum_r g(r)\top \theta)}{\sum_{y' \in \text{GEN}(x)} \exp(\sum_{r' \in y'} g(r')\top \theta)} \]
Decoding

\[ p_\theta(y \mid x) = \frac{\exp(f(x, y)^\top \theta)}{\sum_{y' \in \text{GEN}(x)} \exp(f(x, y')^\top \theta)} \]
\[ = \frac{\exp(\sum_r g(r)^\top \theta)}{\sum_{y' \in \text{GEN}(x)} \exp(\sum_{r' \in y'} g(r')^\top \theta)} \]

- Assume we have learned the parameters
  - It is enough to find \( y \) that maximizes the nominator
  - Because features decompose we can use CKY
  - Define the potential function:

\[ \Psi(r) = \exp(g(r)^\top \theta), \quad \Psi(y) = \prod_{r \in y} \Psi(r) = \exp(\sum_{r \in y} g(r)^\top \theta) \]
CKY works

• just replace q with $\psi$

**Input:** sentence $s = w_1 \ldots w_n$, A CFG $G = (V, \Sigma, S, R)$

**Initialization:**

for all $A \in N, i \in 1 \ldots n$

$$\pi(i, i, A) = \begin{cases} \Psi(x, A, i) & (A \rightarrow i) \in R \\ 0 & \text{otherwise} \end{cases}$$

**Algorithm:**

for all $i \in 1 \ldots (n - 1), l \in 1 \ldots (n - l)$

$j = i + l$

for all $A \in N,$

$$\pi(i, j, A) = \max_{(A \rightarrow BC) \in R} \max_{s \in i \ldots (j - 1)} \Psi(x, A, B, C, i, j, k) \times \pi(i, s, B) \times \pi(s + 1, j, C)$$

$$\text{bp}(i, j, A) = \arg \max_{(A \rightarrow BC) \in R} \max_{s \in i \ldots (j - 1)} \Psi(x, A, B, C, i, j, k) \times \pi(i, s, B) \times \pi(s + 1, j, C)$$
Learning

$$L(\theta) = \sum_i \log p_\theta(y_i \mid x_i)$$

$$\nabla L(\theta)_i = f(x_i, y_i) - \sum_{y'} p_\theta(y' \mid x)f(x, y')$$

How to compute the second term?
\[
\sum_y \rho_\theta(y \mid x) f(x, y) = \sum_y \sum_r \rho_\theta(y \mid x) g(x, r)
\]

\[
= \sum_{r=(A,B,C,i,j,k)} \sum_y \rho_\theta(y \mid x) g(x, r)
\]

\[
= \sum_{r=(A,B,C,i,j,k)} g(x, r) \sum_{y: r \in y} \rho_\theta(y \mid x)
\]

\[
= \sum_{r=(A,B,C,i,j,k)} g(x, r) q(x, r)
\]
Learning

• If we compute the $q$ terms efficiently then we can compute the gradients efficiently and learn

• The $q$ terms are computed with a dynamic programming algorithm called *inside-outside*.

• Also used in unsupervised parameter estimation for PCFGs

• Similar to CKY (sum instead of max)
Inside-outside algorithm

- **Input:**
  - sentence $x=x_1...x_n$
  - A CFG $G$
  - A potential function

  $$\Psi(r) = \exp(g(r)^\top \theta), \quad \Psi(y) = \prod_{r \in y} \Psi(r) = \exp\left(\sum_{r \in y} g(r)^\top \theta\right)$$

- **Output:**

  $$Z = \sum_{y \in \text{GEN}(x)} \Psi(y)$$

  $$\mu(r) = \sum_{y : r \in y} \Psi(y)$$

  Note that $q(r) = \frac{\mu(r)}{Z}$
Overall idea

- In forward-backward, we computed the probability until a certain position $i$ with tag $t$ and from position $i$ with tag $t$ until the end of the sequence (prefix and suffix)

```
I ate the large cake
```

- forward    backward
Overall idea

• Here the analog will be inside tree score and outside tree score
• We will compute those for every non-terminal and span

```
NP
/   \                  /   /
DT    NN              NP  VP
   |                     NP
   |         The dog saw
   |         NP
   |         PP
   |         IN
   |         in
   |         DT
   |         the
   |         NP
   |         the park
```

```
NP
/   /
DT    NN
   |      VBD
   |      NP
   |      PP
   |      IN
   |      in
   |      DT
   |      the
   |      NP
   |      the park
```

\( x = \)
Inside scores

• Let $Y(A, i, j)$ be the set of trees (over $x$) with root $A$ that span $x_i \ldots x_j$.

• We will compute with dynamic programming the sum of potentials for all trees in $Y(A, i, j)$

$$\alpha(A, i, j) = \sum_{y \in Y(A, i, j)} \Psi(y)$$
Outside scores

- Let $O(A, i, j)$ be the set of outside trees with non-terminal $A$ over span $x_i \ldots x_j$.

- An outside tree is:
  - Rooted in $S$
  - Uses rules from the CFG
  - Yields $x_1 \ldots x_{i-1}Ax_{j+1} \ldots x_n$
Outside tree example

*The dog saw the man in the park*

O(NP, 4, 5)
Outside scores

• We compute with dynamic programming the sum of potentials of all outside trees for every non-terminal and span:

\[
\beta(A, i, j) = \sum_{y \in \mathcal{O}(A, i, j)} \Psi(y)
\]
Computing marginals given inside and outside scores

\[ Z = \sum_{y \in \mathcal{Y}} \Psi(y) = \alpha(S, 1, n) \]

Let \( t^i_{A,i,j}, t^o_{A,i,j} \) be the inside tree and outside tree for a span \( x_i \ldots x_j \) with non-terminal \( A \) for any tree \( y \) containing rule production \( r = (A, B, C, i, j, k) \):

\[ \Psi(y) = \Psi(t^o_{A,i,j}) \times \Psi(r) \times \Psi(t^i_{B,i,k}) \times \Psi(t^i_{C,k+1,j}) \]

It follow that:

\[ \mu(r) = \sum_{y: r \in y} \Psi(y) = \sum_{t_1 \in \mathcal{O}(A,i,j)} \sum_{t_2 \in \mathcal{T}(B,i,k)} \sum_{t_3 \in \mathcal{T}(C,k,j)} \Psi(t_1) \times \Psi(r) \times \Psi(t_2) \times \Psi(t_3) \]

\[ = \Psi(r) \times \left( \sum_{t_1 \in \mathcal{O}(A,i,j)} \Psi(t_1) \right) \times \left( \sum_{t_2 \in \mathcal{T}(B,i,k)} \Psi(t_2) \right) \times \left( \sum_{t_3 \in \mathcal{T}(C,k,j)} \Psi(t_3) \right) \]

\[ = \Psi(r) \times \beta(A, i, j) \times \alpha(B, i, k) \times \alpha(C, k, j) \]

Similarly it is simple to show that \( \mu(A, i) = \alpha(A, i, i) \times \beta(A, i, i) \)
Computing inside scores

- Computing inside scores recursively:

**Initialization:**
For all $i \in 1 \ldots n$, for all $A \in N$:

$$\alpha(A, i, i) = \Psi(A, i) \text{ if } (A \rightarrow x_i \in R), \ 0 \text{ otherwise}$$

**Recursion:**
For all $1 \leq i < j \leq n$, for all $A \in N$:

$$\alpha(A, i, j) = \sum_{A \rightarrow B} \sum_{C \text{ } k=i}^{j-1} (\Psi(A, B, C, i, j, k) \times \alpha(B, i, k) \times \alpha(C, k, j))$$
Computing outside scores

Initialization:
For all $A \in N$:

$$\beta(A, 1, n) = 1 \text{ if } A = S, \ 0 \text{ otherwise}$$

Recursion:
For all $1 \leq i \leq j \leq n, (i, j) \neq (1, n)$, for all $A \in N$:

$$\beta(A, i, j) = \sum_{B \rightarrow C} \sum_{A k=1}^{i-1} (\Psi(B, C, A, k, i - 1, j) \times \alpha(C, k, i - 1) \times \beta(B, k, j))$$

$$+ \sum_{B \rightarrow A} \sum_{C k=j+1}^{n} (\Psi(B, A, C, i, j, k) \times \alpha(C, j + 1, k) \times \beta(B, i, k))$$
Remaining details

- Complexity:
  - To compute both inside and outside scores we again go over every span and every split point and sum over all rules and split points
  - $O(n^3|N|^3)$

- Evaluation (with work on feature engineering):
  - 90.5 F1

- Finkel et al., 2008, Hall et al., 2014, …
Summary

• To use a global linear model (CRF) for parsing
  • Features need to decompose
  • Use Viterbi for decoding
  • Need to compute rule production marginals for the gradient
    • inside-outside algorithm
  • Complexity is the same as Viterbi
Deep models for syntactic parsing
Greedy parsing

- Replace linear scoring function with non-linear
- Tree-LSTMs
Neural CRF
Non-linear scoring function

- The only requirement for CKY and Inside-Outside was that the **global** score of a parse tree decomposes to **local** scores over rule productions:

\[
\Psi(x, y) = \prod_{r \in y} \Psi(x, r)
\]

- We used a linear score for \(\psi\), but it can be a non-linear neural network, and the decoding would remain unchanged and correct

- We will now see such a neural network
Assume all features are conjunctions of “rule features” and “span” features

\[ \psi_{\text{sparse}}(x, A, B, C, i, j, k) = f_s(x, i, j, k)^\top W f_o(A, B, C) \]

Weight for conjunction of rule features and span feature
Learned representation

- Replace span features by a neural network
- We learn surface representations

\[ f_w(x, i, j, k) : \text{fixed-length sequence of word indicators} \]
\[ v(f_w) : \text{embedding of } f_w \]
\[ \psi_{\text{neural}}(x, A, B, C, i, j, k) = \text{relu}(H \cdot v(f_w))^\top W f_o(A, B, C) \]
Final representation

\[ \Psi(\cdot) = \Psi_{\text{sparse}}(\cdot) + \Psi_{\text{neural}}(\cdot) \]
Features

- $f_w$: reflected the side of personality.
- $v(f_w)$:
- $f_s$: [[PreviousWord = reflected]], [[SpanLength = 7]], …
Learning and inference

- SGD

- Can compute inside scores or feature expectations with the inside-outside algorithm in the neural network.

- Inference uses standard CKY after computing all of the local scores

- Evaluation: 91.3 F1
Minimal span-based parser

• Have a BiLSTM encode the sentence

• Represent span (i, j) by concatenating (f_i - f_j) and (b_i-b_j)

• Score labeled spans with s\text{label}(i, j, l)+ s(i,j)

• Find the tree with highest score with CKY

• But show also a greedy top-down parsing performs great!

• 91.8 F1
Minimal span-based parser
RNN grammars

• Top-down parsing algorithms

<table>
<thead>
<tr>
<th>Stackₜ</th>
<th>Bufferₜ</th>
<th>Open NTsₜ</th>
<th>Action</th>
<th>Stackₜ₊₁</th>
<th>Bufferₜ₊₁</th>
<th>Open NTsₜ₊₁</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>B</td>
<td>n</td>
<td>NT(X)</td>
<td>S</td>
<td>(X)</td>
<td>B</td>
</tr>
<tr>
<td>S</td>
<td>x</td>
<td>B</td>
<td>SHIFT</td>
<td>S</td>
<td>x</td>
<td>B</td>
</tr>
<tr>
<td>S</td>
<td>(X</td>
<td>τ₁</td>
<td>…</td>
<td>τₖ)</td>
<td>B</td>
<td>n</td>
</tr>
</tbody>
</table>

Input: The hungry cat meows.

<table>
<thead>
<tr>
<th>Stack</th>
<th>Buffer</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>(S)</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>(S</td>
<td>NP)</td>
</tr>
<tr>
<td>3</td>
<td>(S</td>
<td>NP</td>
</tr>
<tr>
<td>4</td>
<td>(S</td>
<td>NP</td>
</tr>
<tr>
<td>5</td>
<td>(S</td>
<td>NP</td>
</tr>
<tr>
<td>6</td>
<td>(S</td>
<td>NP</td>
</tr>
<tr>
<td>7</td>
<td>(S</td>
<td>NP</td>
</tr>
<tr>
<td>8</td>
<td>(S</td>
<td>NP</td>
</tr>
<tr>
<td>9</td>
<td>(S</td>
<td>NP</td>
</tr>
<tr>
<td>10</td>
<td>(S</td>
<td>NP</td>
</tr>
<tr>
<td>11</td>
<td>(S</td>
<td>NP</td>
</tr>
</tbody>
</table>
RNN Grammars Model
RNN Grammars Learning and Inference

- Learning: ML
- Inference: Approximate (sampling-based)
- Results: 92.4 when running generative re-ranking of discriminative parser

\[ p(x, y) = \prod_{t=1}^{|a(x,y)|} p(a_t \mid a_{<t}) \]

\[ = \prod_{t=1}^{|a(x,y)|} \frac{\exp r_{a_t}^\top u_t + b_{a_t}}{\sum_{a' \in A_G(T_t,S_t,n_t)} \exp r_{a'}^\top u_t + b_{a'}} \]

\[ u_t = \tanh(W[o_t; s_t; h_t] + c) \]
State-of-the-art

• 95.6 by doing span-based parsing + BERT pre-training

• Training with **loss-augmented** hinge loss

\[
s(T) = \sum_{(i,j,l) \in T} s(i, j, l)
\]

\[
\hat{T} = \arg \max_T s(T)
\]

\[
J(\theta) = \max(0, \max_{T \neq T^*} [s(T) + \Delta(T, T^*)] - s(T^*))
\]