Syntactic analysis

Marco Kuhlmann
Department of Computer and Information Science
Syntactic analysis

- **Syntactic analysis** or **syntactic parsing** is the task to map a sentence to a formal representation of its syntactic structure.

- The syntactic structure of a sentence provides important clues about the meaning of the sentence.
Different syntactic representations

Phrase structure tree

Dependency tree

Source: Wikimedia Commons [1] [2]

Noam Chomsky

Lucien Tesnière
A dependency tree for an $n$-word sentence $x$ is a directed graph $G = (V, A)$ where $V = \{0, \ldots, n\}$ and where every vertex $v \in V$ is reachable from the vertex 0 via exactly one directed path.

The vertex 0 is called the root vertex of the dependency tree $G$. 
This Stanford University alumnus co-founded educational technology company Coursera.

SPARQL query against DBPedia

```
SELECT DISTINCT ?x WHERE {
  dbr:Coursera dbo:foundedBy ?x.
}
```
Syntactic structure, semantic relations

Subject: Koller  co-founded: Coursera

dbo:Coursera dbo:foundedBy dbr:Daphne_Koller
Representation of dependency trees

The tree is represented by the list of its head positions.
Every subtree corresponds to a contiguous sequence of words.
Non-projective dependency trees

The sequence of words in a subtree may contain ‘gaps’.

omdat Jan Piet Marie zag helpen lezen
Number of projective dependency trees

Source: http://oeis.org/A006013
Algorithmic approaches

- **Exhaustive search**
  
  Cast parsing as a combinatorial optimisation problem over the set of target representations (projective dependency trees).

  *Eisner algorithm*

- **Greedy search**
  
  Cast parsing as a sequence of classification problems: at each point in time, predict one of several parser actions.

  *arc-standard algorithm*
Announcing SyntaxNet: The World’s Most Accurate Parser Goes Open Source
Thursday, May 12, 2016

Posted by Slav Petrov, Senior Staff Research Scientist

At Google, we spend a lot of time thinking about how computer systems can read and understand human language in order to process it in intelligent ways. Today, we are excited to share the fruits of our research with the broader community by releasing SyntaxNet, an open-source neural network framework implemented in TensorFlow that provides a foundation for Natural Language Understanding (NLU) systems. Our release includes all the code needed to train new SyntaxNet models on your own data, as well as Parsey McParseface, an English parser that we have trained for you and that you can use to analyze English text.
This lecture

- Introduction to syntactic analysis
- Eisner algorithm
- Arc-standard algorithm
Eisner algorithm
Dependency parsing as combinatorial optimisation

Given a sentence $x$ and a set $Y$ of candidate dependency trees for $x$, we want to find a highest-scoring dependency tree $\hat{y} \in Y$:

$$\hat{y} = \arg \max_{y \in Y} \text{score}(x, y)$$

The computational complexity of this problem will depend on the choice of the set $Y$ and the scoring function.

Here, $Y$ is the set of all projective dependency trees.
The arc-factored model

Under the **arc-factored model**, the score of a dependency tree is the sum of the scores of its arcs:

$$\hat{y} = \arg \max_{y \in Y} \sum_{a \in y} \text{score}(x, a)$$

The score of an arc is defined as the dot product of a feature vector $\phi(x, a)$ and a learned weight vector $\mathbf{w}$:

$$\text{score}(x, a) = \phi(x, a) \cdot \mathbf{w}$$
Collins algorithm and Eisner algorithm

- Collins and Eisner are two algorithms for computing the highest-scoring projective dependency tree under an arc-factored model.

- Both algorithms solve the optimisation problem using bottom-up dynamic programming.

  score of a tree = sum of the scores of smaller subtrees

- The two algorithms differ with respect to the specific strategy and also with respect to their asymptotic runtime.
Collins’ algorithm

- We are given an input sentence $x$ of length $n$ and a table $A$ that holds the score of each possible arc: $A[h][d] = \text{score}(x, h \rightarrow d)$

- We fill a table $T$ such that $T[i][k][r]$ holds the maximal possible score of a projective dependency tree that covers all words from position $i$ to position $k$ and whose root vertex is at position $r$.

- The cell $T[0][n][0]$ will hold the maximal possible score of a projective dependency tree that covers the complete sentence and whose root vertex is at position 0.
Collins’ algorithm: Basic idea

\[ T[i][k][h] \]
Collins’ algorithm: Basic idea

- In the most simple case, we have $i = k = h$. This case corresponds to a tree consisting of a single vertex.
  
  score $= 0$ (no arcs)

- In the general case, we have $i < k$ and $i \leq h \leq k$. In this case we can decompose the tree into two smaller trees.
  
  score $= \text{sum of the scores of the smaller trees} + \text{score of a single arc}$
Collins’ algorithm: Basic idea

$T[i][k][h]$
Collins’ algorithm: Basic idea

\[ A[h][d] \]

\[ h \rightarrow d \]

\[ T[i][j][h] \quad T[j+1][k][d] \]
Collins’ algorithm

foreach i from 0 to n:
    T[i][i][i] = 0

# non-trivial cases, from smallest to largest
foreach k from 1 to n:
    foreach i from k-1 downto 0:
        foreach j from i to k-1:
            foreach r1 from i to j:
                # root of left subtree
                foreach r2 from j+1 to k:
                    # root of right subtree
                    T[i][j][r1] \max= T[i][j][r1] + T[j+1][k][r2] + A[r1][r2]  # r1 -> r2
                    T[i][k][r2] \max= T[i][j][r1] + T[j+1][k][r2] + A[r2][r1]  # r2 -> r1

# trivial case: single-vertex tree
Complexity analysis of Collins’ algorithm

- The space complexity of Collins’ algorithm is in $O(n^3)$; this corresponds to the number of cells in the table $T$.

- The runtime complexity of Collins’ algorithm is in $O(n^5)$; this corresponds to the number of nested for loops that we need to enumerate all possible combinations.
Learning the score matrix

- The scores in the matrix $A$ can be learned from **treebanks**: sentences annotated with their gold-standard dependency trees.

- For this we need a feature function $\phi(x, a)$ that maps a pair of a sentence $x$ and an arc $a$ to a feature vector.
  
  typical features: word forms, part-of-speech tags, length, …

- With this the score of an arc is the dot product of the feature vector and a matching weight vector: $\text{score}(x, a) = \phi(x, a) \cdot w$
Learning algorithm for the structured perceptron

Define the feature vector of a tree $t$ as $\phi(x, t) = \sum_{a \in t} \phi(x, a)$. 

\[ w \leftarrow 0 \]

for each epoch $e$ do

for each training example $(x, y)$ do

\[ p \leftarrow \text{that tree } t \text{ for which } \text{score}(x, t) = \phi(x, t) \cdot w \text{ is maximal} \]

\[ w \leftarrow w - \phi(x, p) \]

\[ w \leftarrow w + \phi(x, y) \]

Here we use the Collins algorithm!
The Eisner algorithm

- We are given an input sentence $x$ of length $n$ and a table $A$ that holds the score of each possible arc: $A[h][d] = \text{score}(x, h \rightarrow d)$

- Instead of just one table, we now fill 4 different tables $T_t$ such that $T_t[i][k]$ holds the maximal possible score of a dependency graph of a certain type $t$, where $i$ and $k$ are positions in the sentence.

- The cell $T_2[0][n]$ will hold the maximal possible score of a projective dependency tree that covers the complete sentence and whose root vertex is at position 0.
Subproblems in the Eisner algorithm

- **Type 1**: Tree with root at position $k$
- **Type 2**: Tree with root at position $i$
- **Type 3**: Pair of trees with arc from $k$ to $i$
- **Type 4**: Pair of trees with arc from $i$ to $k$
Eisner algorithm: Basic idea

\[ T_2[i][k] \]
Eisner algorithm: Basic idea

- In the most simple case, we have \( i = k \).
  This case corresponds to a tree consisting of a single vertex.
  \[
  \text{score} = 0 \text{ (no arcs)}
  \]

- In the general case, we have \( i < k \).
  In this case we can decompose the tree into two smaller parts.
  \[
  \text{score} = \text{sum of the scores of the smaller parts} + \text{score of a single arc}
  \]
Eisner algorithm: Basic idea

\[ T_{2}[i][k] \]
Eisner algorithm: Basic idea

\[ A[i][d] \]

\[ i \rightarrow d \]

\[ T_2[i][j] \quad T_1[j+1][d] \]
Eisner algorithm: Basic idea

\[ T_{4}[i][d] \]
Eisner algorithm: Basic idea

\[ T_4[i][d] \quad T_2[d][k] \]
The Eisner algorithm (fragment)

```
foreach k from 1 to n:  # max
    foreach i from k-1 downto 0:  # min
        foreach j from i to k-1:  # triangle + triangle + arc ->
            T4[i][k] max= T2[i][j] + T1[j+1][k] + A[i][k]
        foreach j from i to k-1:  # triangle + triangle + arc <-
            T3[i][k] max= T2[i][j] + T1[j+1][k] + A[k][i]
        foreach j from i+1 to k:  # box + triangle
            T2[i][k] max= T4[i][j] + T2[j][k]
        foreach j from i to k-1:  # triangle + box
            T1[i][k] max= T1[i][j] + T3[j][k]
```
Complexity analysis of the Eisner algorithm

- The space complexity of the Eisner algorithm is in $O(n^2)$; this corresponds to the number of cells in a table $T_t$.
- The runtime complexity of the Eisner algorithm is in $O(n^3)$; this corresponds to the number of nested for loops that we need to enumerate all possible combinations.
Learning algorithm for the structured perceptron

Define the feature vector of a tree \( t \) as
\[
\phi(x, t) = \sum_{a \in t} \phi(x, a).
\]

\( w \leftarrow 0 \)

**for each epoch \( e \) do**

**for each** training example \((x, y)\) **do**

\[
p \leftarrow \text{that tree } t \text{ for which } \text{score}(x, t) = \phi(x, t) \cdot w \text{ is maximal}
\]

\[
w \leftarrow w - \phi(x, p)
\]

\[
w \leftarrow w + \phi(x, y)
\]

We can also use the Eisner algorithm!
This lecture

- Introduction to syntactic analysis
- Eisner algorithm
- Arc-standard algorithm
Arc-standard algorithm
Dependency parsing as classification

- In Section 3 we have seen how part-of-speech tagging can be broken down into a sequence of classification problems.
  part-of-speech tagging with the averaged perceptron, with a neural network

- In this section we will see how the same idea can be applied to dependency parsing.

- Instead of POS tags, the classifier will predict transitions that take the parser from one configuration to another.
  other names: moves (for transitions), states (for configurations)
Announcing SyntaxNet: The World’s Most Accurate Parser Goes Open Source

Thursday, May 12, 2016

Posted by Slav Petrov, Senior Staff Research Scientist

At Google, we spend a lot of time thinking about how computer systems can read and understand human language in order to process it in intelligent ways. Today, we are excited to share the fruits of our research with the broader community by releasing SyntaxNet, an open-source neural network framework implemented in TensorFlow that provides a foundation for Natural Language Understanding (NLU) systems. Our release includes all the code needed to train new SyntaxNet models on your own data, as well as Parsey McParseface, an English parser that we have trained for you and that you can use to analyze English text.
Transition-based dependency parsing

- The parser starts in the **initial configuration**.
- It then calls the classifier, which predicts the transition that the parser should make to move to the next configuration.
- This process is repeated until the parser reaches a **terminal configuration**.
Configurations

A parser configuration consists of three parts:

- A **buffer**, which contains those words in the sentence that still need to be processed. Initially, the buffer contains all words.

- A **stack**, which contains those words in the sentence that are currently being processed. Initially, the stack is empty.

- A **partial dependency tree**. Initially, this tree contains all the words of the sentence, but no dependency arcs.
Transitions

- The **shift transition (SH)** removes the frontmost word from the buffer and pushes it to the top of the stack.

- The **left-arc transition (LA)** creates a dependency from the topmost word on the stack to the second-topmost word, and removes the second-topmost word.

- The **right-arc transition (RA)** creates a dependency from the second-topmost word on the stack to the topmost word, and removes the topmost word.
Transition-based dependency parsing, example

I booked a flight from L.A.
Transition-based dependency parsing, example

I booked a flight from L.A.
Transition-based dependency parsing, example

I booked a flight from L.A.
Transition-based dependency parsing, example

I booked a flight from L.A.

booked a flight from L.A.

stack buffer

SH classifier
Transition-based dependency parsing, example

I booked a flight from L.A.
Transition-based dependency parsing, example

I booked a flight from L.A.

booked a flight from L.A.

stack buffer

LA classifier
Transition-based dependency parsing, example

I booked a flight from L.A.

stack      buffer

booked    flight    from    L.A.

classifier

SH
Transition-based dependency parsing, example

I booked a flight from L.A.
Transition-based dependency parsing, example

I booked a flight from L.A.

stack    buffer

booked flight from L.A.
Transition-based dependency parsing, example

I booked a flight from L.A.
Transition-based dependency parsing, example

I booked a flight from L.A.
Transition-based dependency parsing, example

I booked a flight from L.A.

(terminal configuration)
Valid transitions

Valid transitions

- SH is valid if the buffer contains at least one word.
- LA and RA are valid if the stack contains at least two words.
  with the artificial root vertex: at least three words for LA

Valid transition sequence

= every transition in the sequence is valid
Soundness and completeness

- **Soundness**
  
  Every valid transition sequence that starts in the initial configuration and ends in some terminal configuration builds some projective dependency tree.

- **Completeness**
  
  Every projective dependency tree can be built by some valid transition sequence that starts in the initial configuration and ends in some terminal configuration.
Features in transition-based dependency parsing

Features can be defined over

- the next words in the buffer
- the topmost words on the stack
- the partial dependency tree
Training transition-based dependency parsers

- To train a transition-based dependency parser, we need a treebank with gold-standard dependency trees.

- In addition to that, we need an algorithm that tells us the gold-standard transition sequence for a tree in that treebank.

  training oracle
Training oracle

- Choose LA if this creates a correct head–dependent relation, based on the gold-standard tree, and if all dependents of the second-topmost word on the stack have already been assigned.

- Choose RA if this creates a correct head–dependent relation, based on the gold-standard tree, and if all dependents of the topmost word on the stack have already been assigned.

- Otherwise, choose SH.

must always be valid, unless the tree is non-projective
Neural network architecture for parsing

LA

ReLU

word emb.

<table>
<thead>
<tr>
<th>4581</th>
<th>hates</th>
</tr>
</thead>
</table>

| 10435 | Kim   |

| 16327 | broccoli |

| 1 | PROPN |
This lecture

- Introduction to syntactic analysis
- Eisner algorithm
- Arc-standard algorithm