Lecture 2: Constituency Parsing

Tianpei Xie

Jun. 27th., 2022

Contents

1 Concepts

Syntactic parsing is the task of assigning a syntactic structure to a sentence. This chapter focuses on *constituency structures*, those assigned by context-free grammars (CFG). In a sense, it is the process of inferring the **parse tree** behind the text. Parse trees can be used in applications such as

- **grammar checking**: sentence that cannot be parsed may have grammatical errors (or at least be hard to read).
- **semantic analysis**: as intermediate stage of representation
- **question answering**:
- **name entity recognition**:

1.1 Ambiguity in syntatic parsing

The most serious problem baced by syntactic parser is **Ambiguity**.

- **part-of-speech ambiguity** and *part-of-speech disambiguation.* **Words** are ambiguous **have more than one possible part-of-speech** and the goal of part-of-speech disambiguation is to find the correct tag for the situation. For example, ”book” can be a verb (”book that flight”) or a noun (”hand me that book”). ”That” can be a determiner (”Does that flight serve dinner”) or a complementizer (thought that your flight was earlier). The goal of POS-tagging is to resolve these ambiguities, choosing the proper tag for the context.
- **structural ambiguity**. Structural ambiguity occurs when the grammar can assign more than one parse tree to a sentence. Structural ambiguity, appropriately enough, comes in many forms. Two common kinds of ambiguity are **attachment ambiguity** and **coordination ambiguity**.
  - **attachment ambiguity**: if a particular constituent can be attached to the parse tree at more than one place. Various kinds of adverbial phrases are also subject to this kind of ambiguity. e.g. ”We saw the Eiffel Tower flying to Paris”. ”flying to Paris” can be gerundive-VP whose subject is ”the Eiffel Tower” or it can be an adjunct modifying the VP headed by ”saw”.
  - **coordination ambiguity**: Phrases can be conjoined by a conjunction like ”and”. For example, the phrase ”old men and women” can be bracketed as ”[old [men and women]]”, referring to ”old men” and ”old women”, or as ”[old men]” and ”[women]”, in which case it is only the men who are old.

2 CKY parsing

The **Cocke-Kasami-Younger (CKY) algorithm** is the most widely used **dynamic-programming** based approach to syntactic parsing. In dynamic programming, each subproblem represents a parse tree for all the constituents detected in the input.

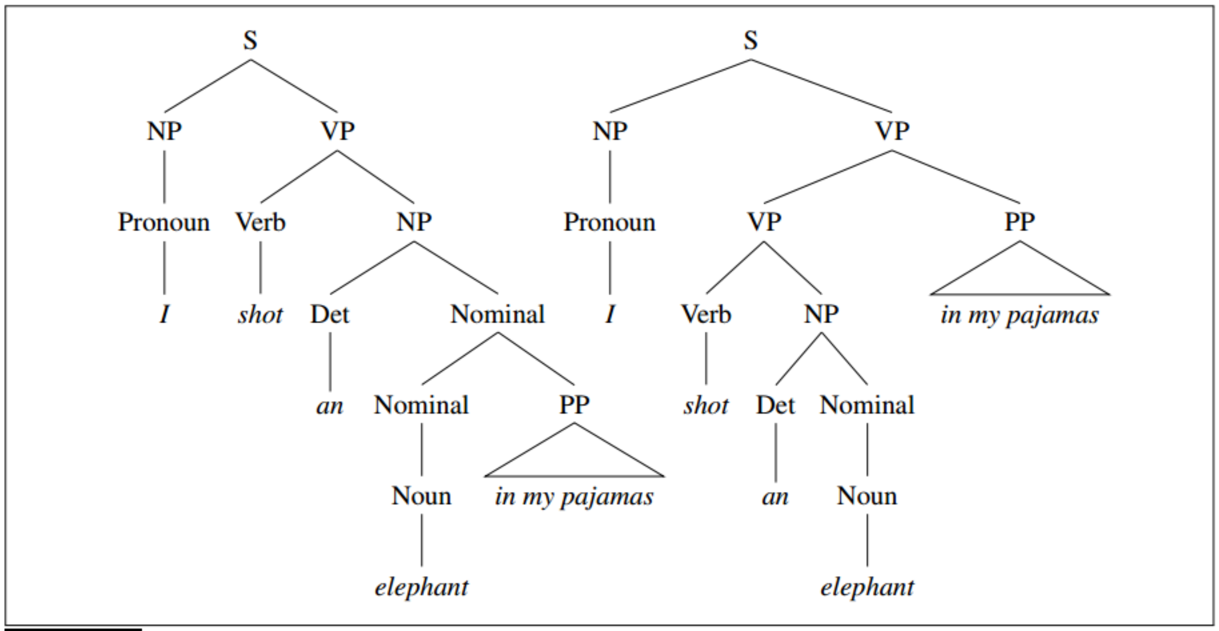


Figure 13.2 Two parse trees for an ambiguous sentence. The parse on the left corresponds to the humorous reading in which the elephant is in the pajamas, the parse on the right corresponds to the reading in which Captain Spaulding did the shooting in his pajamas.

Figure 1: An example structural ambiguity, i.e. the same sentence can be represented by two parse trees.

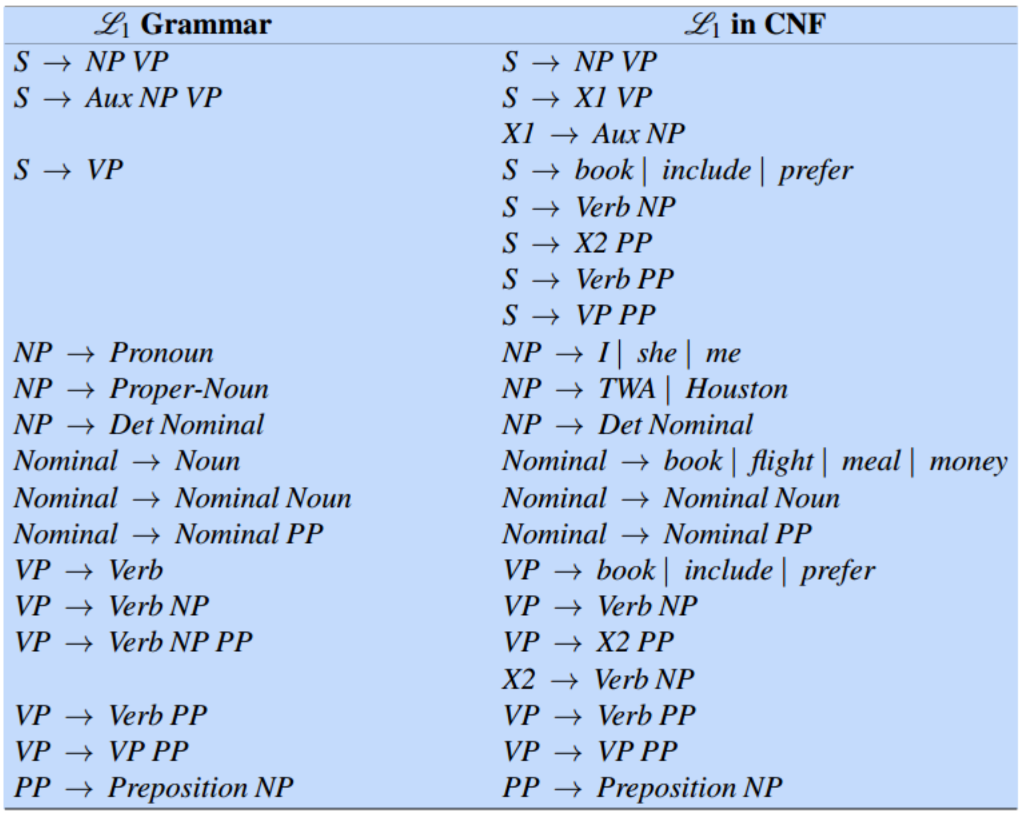
2.1 Conversion to Chomsky Normal Form

The CKY algorithm requires grammars to first be in *Chomsky Normal Form (CNF)*. Note that any context-free grammar can be represented in CNF without lossing information. Lets start with the process of converting a generic CFG into one represented in CNF. Assuming we are dealing with ϵ-free grammar, there are three situations we need to address in any generic grammar:

- rules that **mix terminals with non-terminals** on the right-hand side: simply introduce a **new dummy non-terminal** that covers only the original terminal.
- rules that have **a single non-terminal** on the right-hand side, i.e. **unit productions**. We can eliminate unit productions by rewriting the right-hand side of the original rules with the right-hand side of *all the non-unit production rules* that they ultimately lead to. e.g. if $A \to B$ by a chain of one or more unit productions and $B \to \gamma$ is a non-unit production in our grammar, then we add $A \to \gamma$ for each such rule in the grammar and discard all the intervening unit productions. This leads to flattened rules with a lot of constituents.
- rules in which the length of the right-hand side is **greater than 2**. These are normalized through the introduction of **new non-terminals** that spread the longer sequences over several new rules.

The entire conversion process can be summarized as follows:

1. Copy all conforming rules to the new grammar unchanged.
2. Convert terminals within rules to **dummy non-terminals**.



$\mathscr{L}_1$ **Grammar**	$\mathscr{L}_1$ **in CNF**
S → *NP VP*	*S* → *NP VP*
S → *Aux NP VP*	*S* → *X1 VP*
	X1 → *Aux NP*
S → *VP*	*S* → *book* | *include* | *prefer*
	S → *Verb NP*
	S → *X2 PP*
	S → *Verb PP*
	S → *VP PP*
NP → *Pronoun*	*NP* → *I* | *she* | *me*
NP → *Proper-Noun*	*NP* → *TWA* | *Houston*
NP → *Det Nominal*	*NP* → *Det Nominal*
Nominal → *Noun*	*Nominal* → *book* | *flight* | *meal* | *money*
Nominal → *Nominal Noun*	*Nominal* → *Nominal Noun*
Nominal → *Nominal PP*	*Nominal* → *Nominal PP*
VP → *Verb*	*VP* → *book* | *include* | *prefer*
VP → *Verb NP*	*VP* → *Verb NP*
VP → *Verb NP PP*	*VP* → *X2 PP*
	X2 → *Verb NP*
VP → *Verb PP*	*VP* → *Verb PP*
VP → *VP PP*	*VP* → *VP PP*
PP → *Preposition NP*	*PP* → *Preposition NP*

Figure 13.3 $\mathscr{L}_1$ Grammar and its conversion to CNF. Note that although they aren't shown here, all the original lexical entries from $\mathscr{L}_1$ carry over unchanged as well.

Figure 2: An example grammar conversion to CNF.

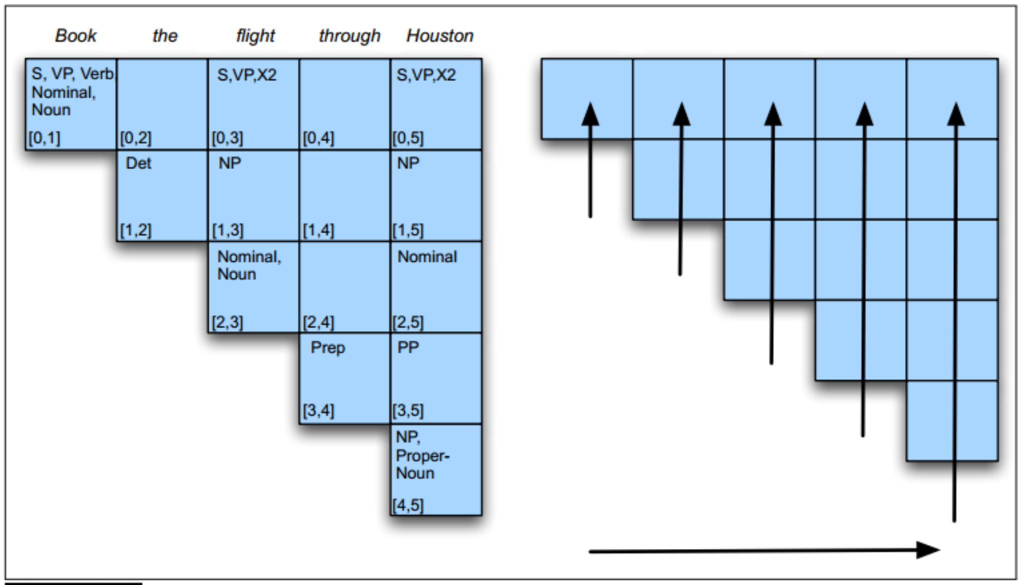


Figure 13.4 Completed parse table for *Book the flight through Houston.*

Figure 3: An example CKY representation.

3. Convert **unit productions**.
4. Make all rules **binary** and add them to new grammar

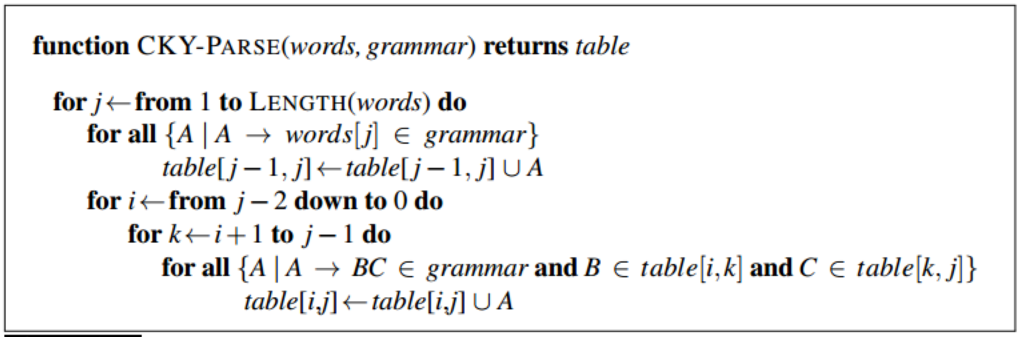
We can see example in Figure 2.

2.2 CKY Recognition

Following CNF, each non-terminal can have exactly **two** children except for the last level. Therefore, we can represent the parse tree in a **two-dimensional matrix**: For a sentence of length n, we will work with the **upper-triangular portion** of an $(n+1) \times (n+1)$ matrix. Each cell $[i, j]$ in this matrix contains the set of *non-terminals* that represent ***all the constituents that span positions*** i ***through*** j of the input. i.e. the syntatic parsed structure for substring $s[i:(j+1)]$. The index can be seen as *fenceposts* "$_0$ Book $_1$ that $_2$ flight $_3$". It follows then that the cell that represents the *entire input* resides in position $[0, n]$ in the matrix.

Note that each non-terminal has two children. So each constituent represented by $[i, j]$ can be decomposed into two parts, i.e. there exists some k, $i < k < j$ so that the constituent can be decomposed into those represented by $[i, k]$ and those represented by $[k, j]$. See example in Figure 3.

Given this reprsentation, the CKY recognition is to fill in the matrix in a bottom-up fashion: before fill in $[i, j]$, we need to fill in all $[i, k], \forall\, k < j$ and $[k, j], \forall\, k > i$, i.e. left to right, bottom to up. The CKY recognition algorithm is described in Figure 4 [Jurafsky and Martin, 2014]. Initialization by fill in the diagonal line with all constituents that maps to the single terminal word at j. The outer loop is over the columns of matrix and the inner loop is over the rows from diagonal upwards. For each intermediate $[i, j]$, search all $k \in [i+1, j-1]$ and fill in A if there exists rule $A \rightarrow B \quad C$, where $B \in [i, k]$ and $C \in [k, j]$, i.e. fill in constituent A that can be split by children constituents B and C from its left side and down side, respectively. Finally, if the $[0, n]$ contains the sentence



```
function CKY-PARSE(words, grammar) returns table

  for $j \leftarrow$ from 1 to LENGTH(words) do
    for all $\{A \mid A \rightarrow words[j] \in grammar\}$
        $table[j-1,j] \leftarrow table[j-1,j] \cup A$
    for $i \leftarrow$ from $j-2$ down to 0 do
      for $k \leftarrow i+1$ to $j-1$ do
        for all $\{A \mid A \rightarrow BC \in grammar$ and $B \in table[i,k]$ and $C \in table[k,j]\}$
            $table[i,j] \leftarrow table[i,j] \cup A$
```

Figure 13.5 The CKY algorithm.

Figure 4: The CKY algorithm.

S, it means that there exists a valid parse tree in this sentence.

Note that Algorithm in Figure 4 is not parser, i.e. it can only **verify** if a valid parse tree exists in the sentence. One can also find the CKY algorithm in book [Skiena, 2020] chap. 8.6.

2.3 CKY Parsing

Algorithm in Figure 4 did not provides the derivation of the sentence, i.e. we do not know how constituents are connected. It turns out that we just need to make some simple modifications so that it can return the parse structure.

- the first change is to augment the entries in the table so that each **non-terminal** is paired with **pointers** to the table entries from which it was derived, i.e. add links to traverse back from the root to terminals at diagonal position.
- the second change is to permit multiple **versions** of the **same non-terminal** to be entered into the table.

Returning an arbitrary single parse consists of choosing an S from cell $[0, n]$ and then **recursively retrieving its component constituents from the table**. See example in Figure 5.

In practice, the conversion of CNF may be very complicated to implement. Also our parser isnt returning trees that are consistent with the grammar given to us by our friendly syntacticians.

One approach to getting around these problems is to keep enough information around to transform our trees back to the original grammar as a post-processing step of the parse. This is trivial in the case of the transformation used for rules with length greater than 2. Simply deleting the new dummy non-terminals and promoting their daughters restores the original tree. In the case of unit productions, it turns out to be more convenient to alter the basic CKY algorithm to handle them directly than it is to store the information needed to recover the correct trees.

Note that it takes $\mathcal{O}(n^2)$ space complexity and $\mathcal{O}(n^3)$ time complexity to complete parsing, i.e. span though matrix cells takes $\mathcal{O}(n^2)$ and for each cell, need to scan through all possible split point k which takes $\mathcal{O}(n)$.

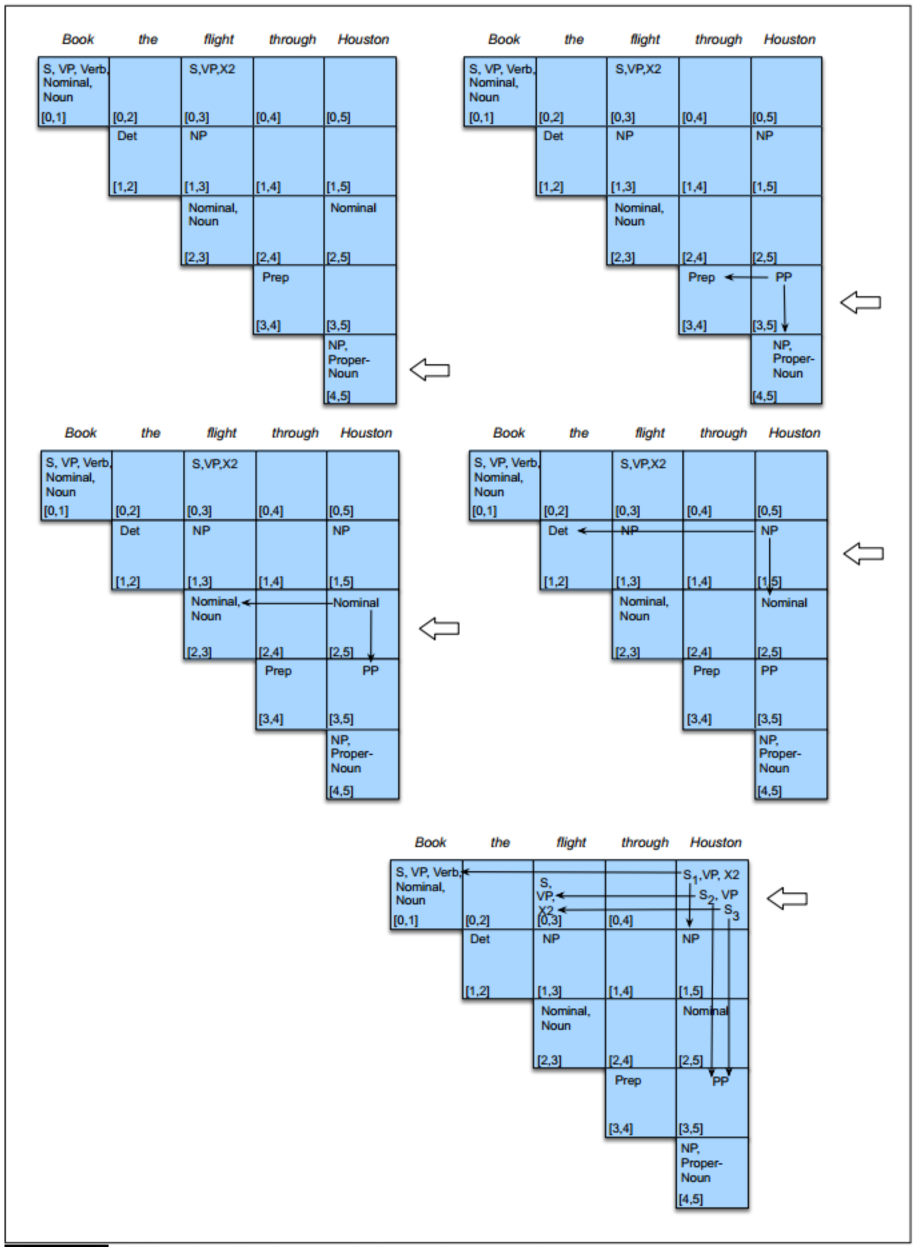


Figure 13.7 Filling the cells of column 5 after reading the word *Houston*.

Figure 5: The example of CKY parsing.

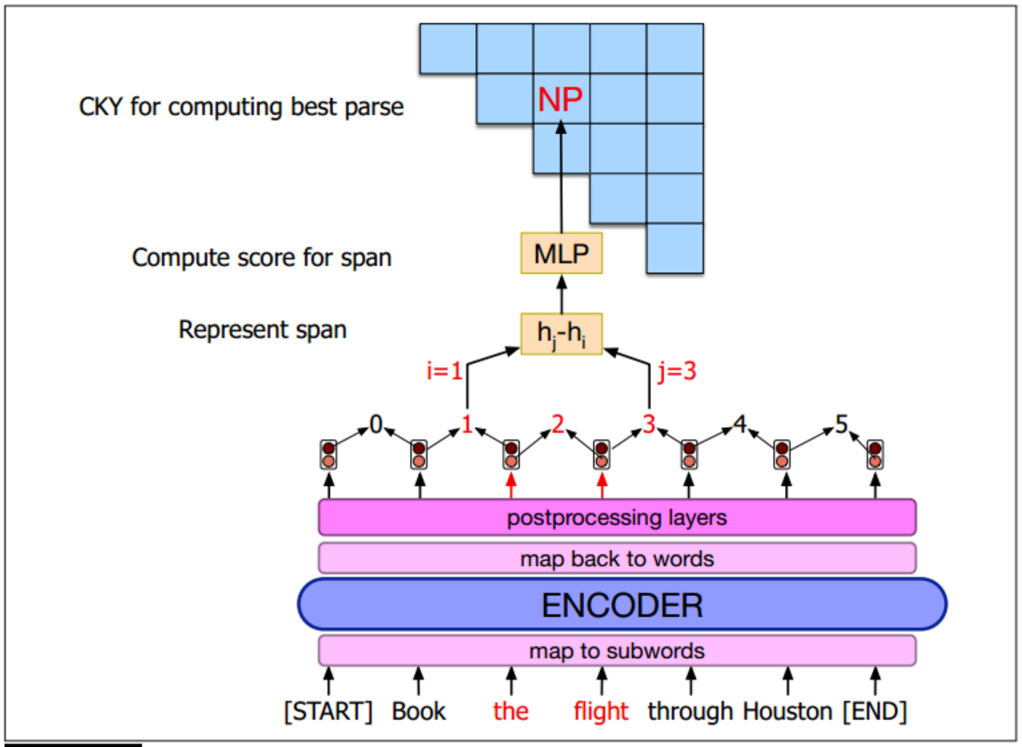


Figure 13.8 A simplified outline of computing the span score for the span *the flight* with the label NP.

Figure 6: The architecture of neural CKY parsing. A score is computed via encoding and mapping of neural network and it is assigned in the matrix of CKY. Then it uses the CKY algorithm to retrieve the optimal path from the root. .

3 Neural CKY parsing

The CKY algorithm doesn't **disambiguate** among the possible parses, i.e. it does not tell which parse tree is correct. To solve the disambiguation problem well use a simple neural extension of the CKY algorithm [Kitaev and Klein, 2018]. The intuition of such parsing algorithms (often called **span-based constituency parsing**, or **neural CKY**), is to train a neural classifier to *assign a score to each constituent*, and then use a modified version of CKY to **combine these constituent scores** to find the *best*-scoring parse tree.

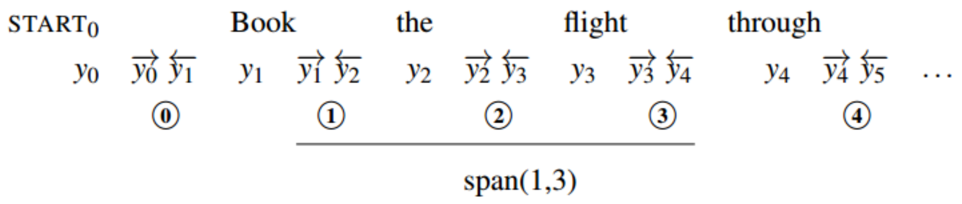

START_0
Book
the
flight
through
y_0 $\overrightarrow{y_0}$ $\overleftarrow{y_1}$ y_1 $\overrightarrow{y_1}$ $\overleftarrow{y_2}$ y_2 $\overrightarrow{y_2}$ $\overleftarrow{y_3}$ y_3 $\overrightarrow{y_3}$ $\overleftarrow{y_4}$ y_4 $\overrightarrow{y_4}$ $\overleftarrow{y_5}$ …
0
1
2
3
4
span(1,3)

Figure 7: The example of scan representation. Each fencepoint has two variables, one scan towards left and one scan towards right.

3.1 Computing scores for a Span

Consider the constituents lie between *words*$[i:j]$ with non-terminal label l, referred as ***span***. The goal for the neural network is to *map the span to a score* $s(i,j,l)$. Figure 6 shows the architecture of neural CKY algorithm. A transformer-based model such as BERT is used to encode the words (sub-words) into embedding representation. (we can map the subword representation back to word representation by looking at the first subword or last subword). The post-processing layers is used after that.

The important step for score computation is to map the word embedding to ***scan embedding***. Note that for each position j, the scan towards left direction may contain different information from the scan towards right direction. So the **key** is to encode two values for each position j, i.e. define $\overrightarrow{y}_j$ as scan beginning with j towards right and $\overleftarrow{y}_j$ as scan beginning with j towards left. See Figure 7. Then the vector **representation** of scan is computed via difference of embeddings between *start* word and *end* word. It is concatenated for two directions.

$$v(i,j) = [\overrightarrow{y}_j - \overrightarrow{y}_i | \overleftarrow{y}_{j+1} - \overleftarrow{y}_{i+1}]$$

The scan vector is then passing through a MLP to be converted into a score

$$s(i,j,l) = \boldsymbol{W}_{2,l}\sigma\left(\text{LayerNorm}\left(\boldsymbol{W}_{1,l}v(i,j)\right)\right)$$

$\sigma(\cdot)$ is the ReLU function. The MLP then outputs a score for each possible non-terminal.

3.2 Computing scores for a parse tree

We can represent a parse tree via the scans. Let $T \in \mathcal{T}$ be a parse tree,

$$T = \{(i_t, j_t, l_t) : \quad t = 1, 2, \ldots, |T|\}.$$

This tree is constructed by retrieving the path from root in the CKY matrix. The score for the parse tree can be obtained by computing the sum of all edges in the tree.

$$s(T) = \sum_{(i_t,j_t,l_t)\in T} s(i_t, j_t, l_t)$$

Thus the optimal tree is the one returns maximum scores, $T^* = \text{argmax}_{T\in\mathcal{T}}\, s(T)$.

This optimal tree can be found via a variant of CKY algorithm [Gaddy et al., 2018]. Here we make some modification of CKY algorithm

- Define $s_{best}(i,j)$ as the optimal score for sub-tree scanning between (i,j)
- It is easy to compute single-word optimal score i.e. **diagnoal entries**, i.e. via scanning through all possible non-terminals

$$s_{best}(i,i+1) = \max_l s(i,i+1,l)$$

- for any (i,j), the following DP-recursion exists:

$$s_{best}(i,i+1) = \max_{k\in[i+1,j-1]} [s_{best}(i,k) + s_{best}(k,j)] + \max_l s(i,j,l).$$

  This has two steps: retrieve best path before it and choose the best current step.

The role of the grammar in classical parsing is to help constrain possible combinations of constituents (NPs like to be followed by VPs). By contrast, the neural model seems to **learn these kinds of contextual constraints** during its mapping from spans to non-terminals.

References

David Gaddy, Mitchell Stern, and Dan Klein. Whats going on in neural constituency parsers? an analysis. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 999–1010, 2018.

Dan Jurafsky and James H Martin. Speech and language processing. vol. 3. *US: Prentice Hall*, 2014.

Nikita Kitaev and Dan Klein. Constituency parsing with a self-attentive encoder. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2676–2686, 2018.

Steven S Skiena. The algorithm design manual (texts in computer science). *group*, 23:2516, 2020.