1 Adding probabilities to rules

- A sentence may have many possible structures: how do we know if one is better than another?
- Weighted CFG: productions are of the form
  \[ X \overset{p}{\rightarrow} \alpha \]
  or equivalently, we write
  \[ P(X \rightarrow \alpha | X) = p \]
  or simply
  \[ P(X \rightarrow \alpha) = p \]
- In a (proper) probabilistic CFG, the weights should be assigned such that
  \[ \sum_{\alpha \text{ s.t. } X \rightarrow \alpha \in G} P(X \rightarrow \alpha) = 1 \]
- The probability of a derivation is the product of the probabilities of the rules used in the derivation (if a rule is used more than once, we count it more than once).

Digression: consistent and inconsistent PCFGs Is it the case that \( \sum P(\text{tree}) = 1 \)? In general, no!

\[
\begin{align*}
S & \overset{0.9}{\rightarrow} SS \\
S & \overset{0.1}{\rightarrow} a 
\end{align*}
\]

In this grammar, the first rewrite rule increases the number of Ss by one, and the second decreases it by one. Since the first is more likely than the second, there is a nonzero chance of never finishing (i.e., deriving an infinite tree!).

Computing the best probability

- Instead of storing a set of nonterminals in each cell, store a mapping from nonterminals to probabilities: $chart[i,j](X) = p$.
- In CKY, initialize $chart[i−1,i](X)$ to $P(X \rightarrow w_i)$.
- And when we recognize $X$ spanning from $i$ to $j$ from $chart[i,k](Y)$ and $chart[k,j](Z)$, we set $chart[i,j](X) = chart[i,k](Y) \times chart[k,j](Z) \times P(X \rightarrow YZ)$.
- As before, no nonterminal can appear twice in a cell. If we want to set $chart[i,j](X) = p'$ but it already has a value $p$, then we set $chart[i,j](X) = \max\{p,p'\}$.
- Then $chart[0,n](S)$ is the best probability.

Why CKY is convenient

Last week we saw two different versions of the CKY algorithm: the original version and the deductive-logic version. The two have identical search spaces, but the latter uses a general-purpose search algorithm that may explore the space in a different order from the original CKY. Why is this bad? Suppose the following happens:

1. We prove $[NP,0,4]$ with probability 0.5 and $[VP,4,10]$ with probability 0.4.
2. We prove $[S,0,10]$ with probability 0.2 from those two items.
3. We prove $[NP,0,4]$ again, but this time with probability 0.6, replacing the 0.5.

But now our probability of 0.2 for $[S,0,10]$ is not optimal. The original CKY makes sure this never happens by always finishing shorter spans before longer spans, and because of Chomsky normal form, never generates an item from another item with the same-size span.

But we still like deductive parsing

- Nevertheless the deductive parser notation is still a faithful representation of CKY’s search space. We can keep the notation and still use the original CKY search algorithm.
- This is handy for describing different computations on derivations:

Axiom $[X, i, i + 1] : p \quad (X \xrightarrow{p} w_{i+1}) \in G$

Inference rule $[Y, i, k] : p_1 \quad [Z, k, j] : p_2 \quad \frac{X, i, j : pp_1p_2}{(X \xrightarrow{p} YZ) \in G}$

Goal $[S, 0, n]$ 

In addition, we specify what to do when an item gets more than one value. Before, it was maximization; now, we abstract away from that to a ‘merge’ operator $\oplus$. For finding the best probability, we used

$$p \oplus p' = \max\{p, p'\}$$
• But, for example, what if we want to compute the total probability of all derivations of the input string? Just redefine

\[ p \oplus p' = p + p \]

This is known as the Inside algorithm.

2 Computing the best structure

• Instead of mapping nonterminals to probabilities \( p \), map them to pairs \((p, T)\), where \( T \) is a (partial) tree.

\[
\text{Axiom} \quad [X, i, i + 1]: \left( p, \frac{X}{w_i} \right) \quad (X \xrightarrow{p} w_{i+1}) \in G
\]

\[
[Y, i, k]: \left( p_1, \frac{X}{T_1} \right) \quad [Z, k, j]: \left( p_2, \frac{X}{T_2} \right)
\]

\[
\text{Inference rule} \quad [X, i, j]: \left( pp_1p_2, \frac{X}{T_1 \& T_2} \right) \quad (X \xrightarrow{p} YZ) \in G
\]

Goal \quad \[S, 0, n]\]

\[
\text{Merge} \quad (p, T) \oplus (p', T') = \begin{cases} (p, T) & \text{if } p > p' \\ (p, T) & \text{if } p' > p \end{cases}
\]

• But this is not very efficient: the same subtrees will get stored over and over.

• Instead use back-pointers to \( T_1 \) and \( T_2 \):

\[
[Y, i, k]: (p_1, T_1) \quad [Z, k, j]: (p_2, T_2)
\]

\[
[X, i, j]: \left( pp_1p_2, \frac{X}{\& T_1 \& T_2} \right) \quad (X \xrightarrow{p} YZ) \in G
\]

3 Computing all structures

• In principle, an exponential number of parses; in practice, millions or billions of parses is not unusual. How can we compute all possible structures?
• Let’s forget about the probabilities for now
• Let item values be *multisets of trees*: \( \{T_1, \ldots, T_k\} \).

\[
\begin{align*}
\text{Axiom} & \quad [X, i, i + 1] : \left\{ \frac{X}{w_i} \right\} \quad (X \xrightarrow{P} w_{i+1}) \in G \\
\text{Inference rule} & \quad [Y, i, k] : T_1 \quad [Z, k, j] : T_2 \\
& \quad [X, i, j] : \left\{ \frac{X}{k \in T_1 \land k \in T_2} \right\} \quad (X \xrightarrow{P} YZ) \in G
\end{align*}
\]

\[
\text{Goal} \quad [S, 0, n]
\]

\[
\text{Merge} \quad T \oplus T' = T \cup T'
\]

• Note, crucially, that the back-pointers are to whole multisets of trees. The resulting web of tree-edges and back-pointers is called a *shared forest* or *packed forest* because it compactly represents a set of trees.

• Abstractly, it is a weighted *hypergraph* or *and-or graph*.

• We can extract individual trees out of a shared forest, or perform operations on the forest as a whole (for example, a compatible translation model could produce a packed representation of all possible translations of all possible parses).