Recitation notes on coarse-to-fine

Chan Young Park

October 4, 2019

1 Notation

Our notation is similar to the one in the lectures:

- $\beta_{(\cdot)}(X, i, j)$ is the *inside score* of nonterminal X spanning $[i, j)$: it is the sum of scores of all parse sub-trees over $[i, j)$ having X as root.
- $\alpha_{(\cdot)}(X, i, j)$ is the *outside score* of X spanning $[i, j)$: it is the sum of scores of all paths of getting to X as a root of the sub-tree over $[i, j)$ from the outside.
- β_T and β_B correspond to top and bottom inside score charts, α_T and α_B stand for top and bottom outside score charts. As in CKY, top chart is the one where the last applied rule *while going bottom-up* was unary, bottom chart is the one where it was binary.
- s_T and s_B are top and bottom CKY charts.

Note that all the formulas in this note are **not in log space**.

2 Recap: CKY

Remember how we filled in bottom and top score charts in CKY:

$$s_T(X, i, j) = \max_{X \to Y} P(X \to Y) \cdot s_B(Y, i, j) \tag{1}$$

$$s_B(X, i, j) = \max_{X \to YZ} \max_{k \in (i,j)} P(X \to YZ) \cdot s_T(Y, i, k) \cdot s_T(Z, k, j) \tag{2}$$

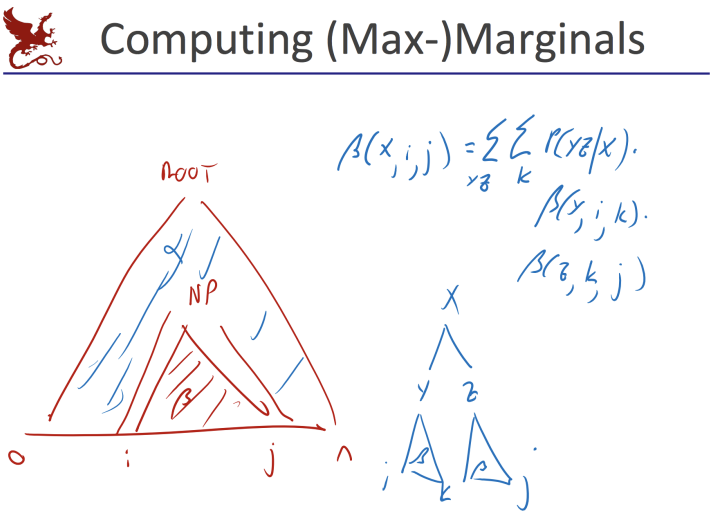

Figure 1: Illustration for notations in this note.

3 Inside scores

Inside scores are computed the same was as regular CKY scores, but with sums instead of maxes. Instead of remembering one best path up to X, you sum over all possible paths. It relates to CKY similarly to how forward algorithm relates to Viterbi.[1]

$$\beta_T(X, i, j) = \sum_{X \to Y} P(X \to Y) \cdot \beta_B(Y, i, j) \tag{3}$$

$$\beta_B(X, i, j) = \sum_{X \to YZ} \sum_{k \in (i,j)} P(X \to YZ) \cdot \beta_T(Y, i, k) \cdot \beta_T(Z, k, j) \tag{4}$$

Inside scores are computed in the same fashion as CKY probabilities:

1. Loop over all possible non-terminals Y in the grammar
2. If you're filling in bottom chart, loop over possible split points k
3. Check if Y is possible over its assigned span ($[i, j)$ for top chart or $[i, k)$ for bottom chart)
4. If it is, loop over closed unary rules of the form $X \to Y$ (top chart) or binary rules of the form $X \to YZ$ (bottom chart).

The only difference is that maxes are now replaced with sums, so instead of comparing the new score to the current one, you add the new score to the existing β.

[1]This analogy is intuitive and helpful, but not mathematically correct, because the conditional probabilities actually flow the other way; remember the explanation of how parsing relates to HMM from Akshay's recitation.

4 Outside scores

You start by setting ROOT as the only possible nonterminal over the whole span:

$$\alpha_T(ROOT, 0, n) = 1$$

One way to compute outside scores is by propagating top-down, indexing rules by parent. First, you check what non-terminals are possible over your span in the top chart, and then consider all possible rules that go from them to the bottom chart:

$$\alpha_B(Y, i, j) \mathrel{+}= P(X \to Y) \cdot \alpha_T(X, i, j) \tag{5}$$

Filling in the top chart is slightly more complicated:

1. Again, you can loop over non-terminals X checking if X is possible over $[i, j)$ in the bottom chart
2. For every feasible X loop over every split point k
3. Loop over rules of the form $X \to YZ$ and increase your top chart values in the following way:

$$\alpha_T(Y, i, k) \mathrel{+}= P(X \to YZ) \cdot \alpha_B(X, i, j) \cdot \beta_T(Z, k, j) \tag{6}$$
$$\alpha_T(Z, k, j) \mathrel{+}= P(X \to YZ) \cdot \alpha_B(X, i, j) \cdot \beta_T(Y, i, k) \tag{7}$$

It's important to understand all the multipliers in the right hand side. $\alpha_B(X, i, j)$ is the outside score of the parent; it's the sum of all paths outside the span $[i, j)$ that have X over this span. $\beta_T(Z, k, j)$ in (6) is the inside score of the right child; if you are summing over all the paths except the ones inside $[i, k)$, you also sum over the right child span $[k, j)$. Similarly, $\beta_T(Y, i, k)$ in (7) is to marginalize over the left child span $[i, k)$.

If you compute a closed-form solution, you will arrive at the same formula as in the lectures:

$$\begin{aligned}\alpha_T(Y, i, j) = &\sum_{X \to ZY} \sum_{l \in [0, i)} P(X \to ZY) \cdot \alpha_B(X, l, j) \cdot \beta_T(Z, l, i) \\ &+ \sum_{X \to YZ} \sum_{r \in [j, n)} P(X \to YZ) \cdot \alpha_B(X, i, r) \cdot \beta_T(Z, j, r)\end{aligned} \tag{8}$$

You should see how this relates to our top-down updates (6) and (7).

5 Pruning

You want to prune the nonterminals for which the marginal probability of being on top of a given span is too small. To decide if a coarse symbol in a top or bottom cell should be pruned, you check if the following inequalities hold:

$$\frac{\beta_T(X,i,j)\alpha_T(X,i,j)}{\beta_T(ROOT,0,n)} > threshold \tag{9}$$

$$\frac{\beta_B(X,i,j)\alpha_B(X,i,j)}{\beta_T(ROOT,0,n)} > threshold \tag{10}$$

Compare this to the marginal probabilities for HMM or CRF (sums of all paths going through a particular node).

6 Algorithm

The algorithm consists of a coarse pass and a fine pass. In the coarse pass, you construct your β_T, β_B, α_T and α_B using the coarse grammar (I used $v = 1, h = 0$). Fine pass is the same as CKY with your fine grammar, but you only update CKY chart cells for nonterminals that are not pruned.

The point of doing this is to reduce the time spent on looping over nonterminals in fine grammar. If in the coarse pass you find that nonterminal A is very unlikely to span $[i, j)$, then in your fine pass there is no point of looping over its fine projections like $A\hat{}B$.

7 Implementation tips

- You should experiment with different values of the pruning threshold. If you set it too low, you will not prune anything and your solution will just become slower. If you set it too high, you will prune some of the correct parses too, so your F1 will go down.
- You should think carefully about how you organize your loops and reduce unnecessary computation. Example: it might be helpful to check if $\beta > 0$ (the node if reachable from the inside) before computing α.

  Coarse-to-fine should make your parser faster, but if you don't think it through, it might get slower instead even if your implementation is correct.
- In the coarse pass, you might want to construct top and bottom boolean charts with pruning masks, having `True` if inequality (9) or (10) respectively holds and `False` otherwise. I can't guarantee this gives you any speedup because my own solution ended up being very suboptimal, but I suggest you think about this on your own.
- Since you will be doing all the calculations in log space, you can use the provided `SloppyMath.logAdd()` function for summing in (3)-(7).