Project 2 Write-up

Maria Ryskina
mryskina@cs.cmu.edu

Abstract

In this project we implemented a CKY parser with and without coarse-to-fine pruning and evaluated it for different configurations of parameters on several sections of Penn TreeBank. We studied how different markovization levels affect the performance of both parsers and performed error analysis.

1 Implementation details

1.1 CKY parsing

For CKY implementation we stored the scores in two separate charts: one where the last rule applied was unary, and another one for binary. While we go up in the chart, we first update the cells on the current row of binary chart based on previous row of unary chart, and then the same row of unary chart based on the binary row that we just did:

$$\text{chartB}[i][j][X] = \max_{X \to YZ} \max_{k \in [i,j)} \text{score}(X \to YZ) \times \\ \times \text{chartU}[i][k][Y] \cdot \text{chartU}[k+1][j][Z]$$

$$\text{chartU}[i][j][X] = \max_{X \to Y} \text{score}(X \to Y) \times \\ \times \text{chartB}[i][j][Y]$$

We do that because we want the last rule applied to be unary (in most cases it is $ROOT \to S$).

We use the lowest level of binary chart to store unary rules of the form $tag \to word$. This is because we want the first rule to be applied to preterminals to also be unary.

We fill the both charts in bottom-to-top, alterating between them and recording back-pointers for the chosen rules (these require two extra charts – for unary and binary rules). To optimize for speed, we go over all the rules by left child (or simply child for unary rules), and only calculate the score if the corresponding cell in the previous chart contains nonzero score.

Then we recursively reconstruct the tree using the saved back-pointers, unroll all unary rules using unary closure, and debinarize the obtained tree.

1.2 Coarse-to-fine pruning

Here we use inside-outside algorithm using coarse grammar ($v = 1, h = 0$) to collect overall probability for each cell and prune those that are too low. For inside scores we go bottom-to-top exactly like in CKY, but replacing maxes with sums:

$$\text{alphaB}[i][j][X] = \sum_{X \to YZ} \sum_{k \in [i,j)} \text{score}(X \to YZ) \times \\ \times \text{alphaU}[i][k][Y] \cdot \text{alphaU}[k+1][j][Z]$$

$$\text{alphaU}[i][j][X] = \sum_{X \to Y} \text{score}(X \to Y) \times \\ \times \text{alphaB}[i][j][Y]$$

For outside scores, we set

$$\text{betaU}[0][n-1][ROOT] = 1$$

and then go top-to-bottom, pushing the scores down. Here the unary chart is the one where we applied a unary rule going bottom-to-top, and same for binary.

$$\text{S} = \text{score}(X \to YZ) \cdot \text{betaB}[i][j][X]$$
$$\text{betaU}[i][k][Y] \mathrel{+}= S \cdot \text{alphaU}[k+1][j][Z]$$
$$\text{betaU}[k+1][j][Z] \mathrel{+}= S \cdot \text{alphaU}[i][k][Y]$$

$$\text{betaB}[i][j][Y] \mathrel{+}= \\ \text{score}(X \to Y) \cdot \text{betaU}[i][j][X]$$

Now we store 4 charts for inside and outside scores; now we compute the full score:

$$\frac{\text{alphaU}[i][j][X] \cdot \text{betaU}[i][j][X]}{\text{alphaU}[0][n-1][ROOT]}$$

	P	R	F1	EX
$v = 2, h = 2$	81.34	79.74	80.53	21.54
$v = \infty, h = 1$	75.24	70.05	72.55	11.08
$v = 1, h = 1$	74.41	67.99	71.06	8.99
$v = 0, h = 1$	64.42	52.43	57.81	3.35

Table 1: CKY without pruning: average precision, recall, F1 and exact count for different grammars.

and

$$\frac{\text{alphaB}[i][j][X] \cdot \text{betaB}[i][j][X]}{\text{alphaU}[0][n-1][ROOT]}$$

and create unary and binary boolean mask charts (having 0 if the corresponding score is lower than the threshold, and 1 otherwise).

Then we perform CKY with fine grammar, with one difference: before updating a cell, we look at the cell of the corresponding coarse symbol in the appropriate mask, and prune this cell if the value is 0.

2 Experimental Evaluation

First, let us compare the performance of CKY for different levels of grammar refinement (horizontal and vertical markovization). The accuracy is shown in Table 2. All experiments were performed on validation set (files 2200 to 2299 of Penn TreeBank) and full dataset ($maxTrainLength = 1000$, $maxTestLength = 40$). The best grammar is $v = 2, h = 2$, and this is the one we are mainly going to use as the fine grammar for coarse-to-fine experiments.

Now let us see how pruning affects the performance. Our initial hypothesis was that it is going to speed the process up but sacrifice a few points in accuracy. The comparison is shown in Table 2.

Accuracy goes down if we increase the threshold, which is understandable: the more we prune, the less potential parse trees we get to compare, so increasing the threshold will decrease of F1. Parsing also becomes faster, because we have to update fewer cells.

However, we still could not beat CKY in terms of time. This must be a defect of this particular implementation: apparently, construction of inside and outside charts for each sentence took too long, and even with a great deal of pruning coarse-to-fine was too slow. Even when we made the pruning strict enough to get a significant decrease in accuracy, it was still slower than simple CKY.

	F1	Time
CKY	80.53	29.42 min
$T = -50$	80.53	46.29 min
$T = -10$	80.43	37.94 min
$T = -5$	75.79	34.77 min

Table 2: Perofrmance of CKY with and without pruning ($v = 2, h = 2$). Threshold values reported in log scale.

	F1	Time
CKY	72.55	45.71 min
$T = -5$	71.61	51.27 min

Table 3: Perofrmance of CKY with and without pruning ($v = 1, h = \infty$). Threshold values reported in log scale.

We also compared results for $v = 1, h = \infty$ grammar: for a complex grammar the coarse pass should be much faster than the fine pass, and then pruning should have a positive effect on time. As you can see in Table 2, that was not the case with our implementation. This is most likely a result of poor optimization in terms of speed on coarse pass.

3 Error Analysis

In this section we will see which types of sentences usually cause our parser to make mistakes. Let us look at our best solution (CKY for $v = 2, h = 2$); the results for coarse-to-fine with the same grammar are very similar.

1. Length is the first major factor. As expected, the longer the sentence is, the harder it becomes for parsing. In our experiments, for all sentences longer than 15 exact match rate is 20% or lower (measured for each length separately).

   To examine this more closely, we look at the whole validation set and calculate average accuracy of CKY for sentences of any particular length. The result is shown on Figure 1. We can see that our parser is very good with short sentences, but as sentences grow, the accuracy decreases.

   We also show a histogram of how the number of errors made by CKY depends on sentence length (Figure 1). It looks like the parser makes the biggest number of mistakes on sentences of length 18-25. However, this is not really a valid measure of quality, because very long and very short sentences are also uncommon in our dataset.

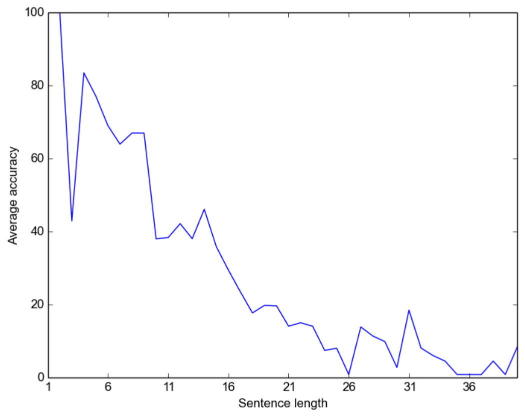

Figure 1: Average accuracy vs. sentence length

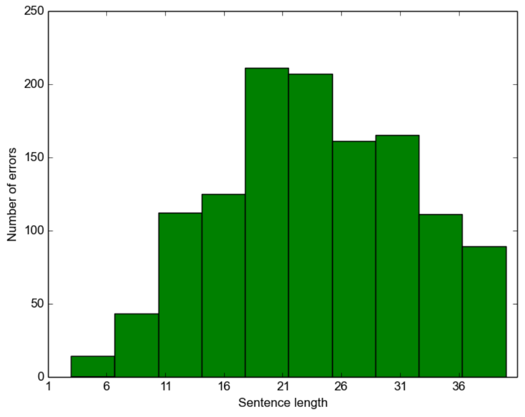

Figure 2: Number of errors vs. sentence length

2. Another measure of structural complexity of a sentence is the occurrence of particular labels in the tree. For example, CKY makes a lot of mistakes on sentences with subordinate clauses (i.e. the gold parse for this sentences contains an $SBAR$ tag). About 86% of trees with $SBAR$ in our validation set are parsed incorrectly, and they constitute 47% of errors CKY makes.

   It is also hard to parse ungrammatical sentence fragments. 14 out of 15 sentences with $FRAG$ get parsed incorrectly, although this is just a minor fraction of our error set.

3. We also had a hypothesis that there might be a correlation between the number of children of $ROOT$ in the gold parse and the error rate. But it turns out that all the trees in our validation set start with a unary rule (in most cases, $ROOT \rightarrow S$ or $ROOT \rightarrow SINV$), so this can not have any effect.

4 Conclusion

We have built a CKY parser with ang without coarse-to-fine and studied how different parameters affect the performance. As expected, pruning decreases accuracy of parsing, but it also turned out to be slower than simple CKY (which is most likely a result of lack of optimization). We have also studied which features of sentences might cause the parser to make mistakes.

In our experiments we tried different levels of grammar complexity and chose the most suitable setting. Our best solution spends about 22 minutes on decoding, and has an F1 score of 80.53 on full validation dataset (84.81 in 15 seconds with $maxTrainLength = maxTestLength = 15$).