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:

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

\[
\text{chartU}_{i,j}[X] = \max_{X \rightarrow Y} \text{score}(X \rightarrow Y) \times \text{chartB}_{i,j}[Y]
\]

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

We use the lowest level of binary chart to store unary rules of the form \(\text{tag} \rightarrow \text{word}\). 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, alternating 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:

\[
\text{alphaB}_{i,j}[X] = \sum_{X \rightarrow Y Z \ k \in [i,j)} \text{score}(X \rightarrow Y Z) \times \alpha_{i,k}[Y] \times \alpha_{k+1,j}[Z]
\]

\[
\text{alphaU}_{i,j}[X] = \sum_{X \rightarrow Y} \text{score}(X \rightarrow Y) \times \alpha_{i,k}[Y]
\]

For outside scores, we set \(\alpha_{0,n-1}[\text{ROOT}] = 1\) 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.

\[
\text{S} = \text{score}(X \rightarrow Y Z) \cdot \text{betaB}_{i,j}[X]
\]

\[
\text{betaU}_{i,k}[Y] \leftarrow \text{S} \cdot \alpha_{i,k}[Y] \cdot \alpha_{k+1,j}[Z]
\]

\[
\text{betaU}_{i,k+1}[Z] \leftarrow \text{S} \cdot \alpha_{i,k}[Y]
\]

\[
\text{betaB}_{i,j}[Y] \leftarrow \text{score}(X \rightarrow Y) \cdot \text{betaU}_{i,j}[X]
\]

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

\[
\frac{\text{alphaU}_{0,n-1}[\text{ROOT}]}{\text{alphaU}_{0,n-1}[\text{ROOT}]}
\]
Table 1: CKY without pruning: average precision, recall, F1 and exact count for different grammars.

<table>
<thead>
<tr>
<th>Grammar</th>
<th>P</th>
<th>R</th>
<th>F1</th>
<th>EX</th>
</tr>
</thead>
<tbody>
<tr>
<td>$v = 2, h = 2$</td>
<td>81.34</td>
<td>79.74</td>
<td>80.53</td>
<td>21.54</td>
</tr>
<tr>
<td>$v = \infty, h = 1$</td>
<td>75.24</td>
<td>70.05</td>
<td>72.55</td>
<td>11.08</td>
</tr>
<tr>
<td>$v = 1, h = 1$</td>
<td>74.41</td>
<td>67.99</td>
<td>71.06</td>
<td>8.99</td>
</tr>
<tr>
<td>$v = 0, h = 1$</td>
<td>64.42</td>
<td>52.43</td>
<td>57.81</td>
<td>3.35</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Threshold</th>
<th>F1</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T = -50$</td>
<td>80.53</td>
<td>29.42 min</td>
</tr>
<tr>
<td>$T = -10$</td>
<td>80.43</td>
<td>46.29 min</td>
</tr>
<tr>
<td>$T = -5$</td>
<td>75.79</td>
<td>37.94 min</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Threshold</th>
<th>F1</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T = -5$</td>
<td>72.55</td>
<td>45.71 min</td>
</tr>
<tr>
<td>$T = -5$</td>
<td>71.61</td>
<td>51.27 min</td>
</tr>
</tbody>
</table>

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.

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, $\text{ROOT} \rightarrow S$ or $\text{ROOT} \rightarrow \text{SINV}$), so this can not have any effect.

4 Conclusion

We have built a CKY parser with and 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 $\text{maxTrainLength} = \text{maxTestLength} = 15$).

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 $\text{SBAR}$ tag). About 86% of trees with $\text{SBAR}$ 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 $\text{FRAG}$ get parsed incorrectly, although this is just a minor fraction of our error set.