Statistical Language Models with Structural Elements

Karl Weilhammer

Institut für Phonetik und Sprachliche Kommunikation
Ludwig-Maximilians-Universität München
Schellingstr. 3
80799 München, Germany
weilkar@phonetik.uni-muenchen.de

ABSTRACT

Statistical $n$-gram language models, which can be seen as a variant of finite state grammars with transition probabilities written on the arcs of the graph, have been applied successfully in speech technology. We will present the new theoretical concept of structural elements and outline an automatic method to improve $n$-gram language models by distributing those structural elements in word sequences. In our model, $n$-grams that have not occurred in the training will be interpolated by $n$-grams containing structural elements. We give a detailed description of the algorithm and present first results of a system trained on a small corpus which consists of spontaneous speech dialogues.

1 INTRODUCTION

The aim of any linguistic theory is to model the structure of a certain language or at least parts of it. Despite the fact that Chomsky (1957) proved that finite state grammars are not powerful enough to model natural language, one variant, the probabilistic $n$-gram models, have been widely applied in speech recognition, automatic translation and many other fields of language technology. This kind of model allows transitions from each word combination of length $(n - 1)$ to all words in the vocabulary and weights them with a probability that can be calculated from the frequency of the respective word combination in a corpus. The problem which arises with this method is that it is impossible to observe all word combinations. Therefore a good strategy for estimating the probability of items that have not occurred in the training has to be developed. A good summary of these techniques is given in Ney, Essen, and Kneser (1994).
In the following, the concept of structural elements (Weilhammer and Ruske, 2000) will be developed. We define a structural element as a symbol that can be placed in a word sequence. It is possible to insert two or more structural elements in a row. If we don’t want to specify whether an item is a structural element or a word, we will use the term text element and the symbol \( \langle A \rangle \). A German dialogue orthographically transcribed in Verbmobil conventions will serve as an example (Burger, 1997), (Wahlster, 2000). Here \( \langle . \rangle, \langle ; \rangle \) and \( \langle ? \rangle \) stand for punctuation marks, \( \langle A \rangle \) denotes breathing and \( \langle P \rangle \) is the marker for a speech pause.

\[
\text{genau } \langle . \rangle \langle A \rangle \langle P \rangle \text{ gut } \langle . \rangle \langle A \rangle \text{ dann ist eigentlich alles in Ordnung } \langle . \rangle \langle A \rangle \text{ sind Sie am Abend schon ausgelastet } \langle . \rangle \text{ oder } \langle A \rangle \text{ haben Sie noch Lust } \langle . \rangle \text{ irgendwas zu unternehmen } \langle ? \rangle
\]

In contrast to the above example, the kind of structural elements that we will use has no a priori meaning. It can be inserted randomly or according to a set of rules. In the following, we will refer to structural elements as \( \langle s \rangle \) and to words as \( w \).

We will describe an algorithm that automatically places structural elements in positions in which they reduce the perplexity of a word sequence. Perplexity is a measure for the average number of possible branches after a word in a sequence. Using the \( n \)-gram approximation the perplexity of a chain of \( T \) text elements is defined in equation (1) (Bahl, Jelinek, and Mercer, 1983). \( W \) is the number of words that are amongst these \( T \) text elements.

\[
PP = \left( \prod_{i=1}^{T-(n-1)} p(t_{i+n}|t_{i+1}...t_{i+n-1}) \right)^{-1/W}
\]

In the following sections we will show how to generate a language model from a sequence of words in which structural elements are optimally placed and how to obtain such a training set. With this model the perplexity of any given word chain can be calculated according to formula (1) by repeated insertion of structural elements between any two words until the minimum of perplexity is found.

2 PROBABILITY DISTRIBUTIONS

To keep explanations and annotations easily comprehensible we will further refer to the case of the bigram language model \( n = 2 \), although all our considerations can directly be generalised to \( n \)-grams with \( n \geq 2 \). For all observed bigrams the probabilities can be calculated from the counts \( N(w_i,w_j) \) and \( N(w_i) \). Bigrams that have not been observed in the training will get zero probability and thus cause infinite perplexity for the whole sequence. We propose two different methods to deal with this case, the back-off model and the rest-element model.

2.1 Back-Off Model

First the simple back-off model will be explained. It removes a constant percentage of the probability mass from each observed bigram and shifts it to all bigrams no matter if observed or not.

\[
p(t_j|t_i) = \begin{cases} \frac{N(w_i,t_j)}{N(t_i)} (1 - D) + D/V \quad & \text{if } N(t_i) > 0 \\ 1/V \quad & \text{if } N(t_i) = 0 \end{cases}
\]
$V$ is the number of words in the vocabulary, $E$ represents the number of different structural elements used and $0 < D < 1$ is the discount value that is necessary to normalise the probability distribution for a fixed $t_x$ to 1.

### 2.2 Rest-Element Model

In the rest-element model, there is one structural element, the rest element $\langle s_R \rangle$, that has transitions with a probability higher than zero to all words. All other bigrams not observed in the training are forbidden and have zero transition probability, regardless of whether they involve two words, or a word and a structural element. Therefore a bigram not observed during the training can always be bridged by two bigrams involving the rest element.

\[
i, j \in \{1, \ldots, V\} \quad x, y \in \{1, \ldots, E - 1\}
\]

\[
p(w_j | w_i) = \frac{N(w_i, w_j)}{N(w_i) + D} \quad \text{(3)}
\]
\[
p(\langle s_y \rangle | w_i) = \frac{N(w_i, \langle s_y \rangle)}{N(w_i) + D} \quad \text{for} \quad y \neq R \quad \text{(4)}
\]
\[
p(\langle s_R \rangle | w_i) = \frac{N(w_i, \langle s_R \rangle) + D}{N(w_i) + D} \quad \text{(5)}
\]
\[
p(w_j | \langle s_x \rangle) = \frac{N(\langle s_x \rangle, w_j)}{N(\langle s_x \rangle) + D} \quad \text{for} \quad x \neq R \quad \text{(6)}
\]
\[
p(w_j | \langle s_R \rangle) = \frac{N(\langle s_R \rangle, w_j) + D}{N(\langle s_R \rangle) + VD} \quad \text{(7)}
\]

Again $0 < D < 1$ is the discount value necessary for the normalisation of the conditional probabilities. For different values of $D$ we get different language models. There is an optimal value for the parameter $D$.

### 3 GENERALISATION AND APPROXIMATION OF UNSEEN BIGRAMS

To demonstrate how a structural element language model works and how the problem of sparse data is solved in this formalism we will use sentence (9) as the training set for a bigram language model. Using the pure maximum likelihood approximation for probabilities without discounting, which involves only the counts of bigram and unigram we get formula (8). On this basis we can derive a set of conditional probabilities (10) that defines a language model. With this language model (10) we can
try to figure out the perplexity of a test sentence (11). Since the bigram \((hit, ball)\) was not observed in the training, its probability is zero and the perplexity goes to infinity.

\[
t_i, t_j \in \{w_1, \ldots, w_V, \langle s_1 \rangle, \ldots, \langle s_E \rangle\} \quad i, j \in \{1, \ldots, V + E\}
\]

\[
p(t_j|t_i) = \begin{cases} 
\frac{N(t_j,t_i)}{N(t_i)} & \text{if } N(t_i) > 0 \\
0 & \text{if } N(t_i) = 0
\end{cases}
\quad \text{(8)}
\]

Training Sentence: \textit{the man hit the ball again} (9)

\[
p(\text{man}|\text{the}) = 0.5 \quad p(\text{hit}|\text{man}) = 1 \quad p(\text{the}|\text{hit}) = 1
\]

\[
p(\text{ball}|\text{the}) = 0.5 \quad p(\text{again}|\text{ball}) = 1
\quad \text{(10)}
\]

Test Sentence: \textit{the ball hit the man} (11)

\[
PP = \left( p(\text{ball}|\text{the}) p(\text{ball}|\text{hit}) p(\text{the}|\text{hit}) p(\text{man}|\text{the}) \right)^{-1/6} = \frac{1}{(0.5 \cdot 0 \cdot 1 \cdot 1)^{1/6}} \to \infty
\]

We will now use the structural element formalism to overcome this problem. Inserting the structural element \(\langle s \rangle\) twice in sentence (9) first before the word \textit{hit} and a second time after the word \textit{ball} results in sentence (12), from which a new language model (13) can be created. Model (13) has transitions involving words and structural elements. They can be used as an approximation for bigrams that did not occur in the training. If we insert \(\langle s \rangle\) between \textit{hit} and \textit{ball} in sentence (11) we get a modified test sentence (14). With the new model (13) and the new test sentence (14) we can substitute \(p(\text{ball}|\text{hit})\) by \(p(\langle s \rangle|\text{hit}) p(\text{ball}|\langle s \rangle)\) and get a perplexity of \(PP = 1.4\).

Training Sentence: \textit{the man \langle s \rangle hit the ball \langle s \rangle again} (12)

\[
p(\text{man}|\text{the}) = 0.5 \quad p(\langle s \rangle|\text{man}) = 1 \quad p(\text{hit}|\langle s \rangle) = 0.5 \quad p(\text{the}|\text{hit}) = 1
\]

\[
p(\text{ball}|\text{the}) = 0.5 \quad p(\langle s \rangle|\text{ball}) = 1 \quad p(\text{again}|\langle s \rangle) = 0.5
\quad \text{(13)}
\]

Test Sentence: \textit{the ball \langle s \rangle hit the man} (14)

On the basis of these considerations we developed an algorithm that finds optimal positions for structural elements by minimising the perplexity on a testset.

4 THE ALGORITHM

We will first explain how to calculate the testset perplexity, since this operation is frequently used in the training procedure. After that we will outline the training algorithm.
### 4.1 Testset Perplexity

Suppose we have a given distribution of structural elements in the training corpus. A language model can be created by counting the $n$-grams and calculating the probabilities $p(t_i | t_{i-n}...t_{i-1})$ for all text elements. After that the perplexity of a second data set (e.g. the testset) can be worked out with a procedure similar to the Viterbi Algorithm (Viterbi, 1967).

**Figure 1:** The set of possible transitions between $w_i$ and $w_{i+1}$ via the structural elements $\langle s_1 \rangle$...$\langle s_N \rangle$ for the calculation of the perplexity of a word sequence.

We calculate the perplexity for the first two text elements. Then we insert a structural element between them and calculate the perplexity again. We do this for all structural elements. After this we keep the best path with the best partial perplexity (see Figure 1), go one step further and continue with the next two text elements until we have reached the last word and worked out the perplexity value of the whole word sequence. After this procedure we have the perplexity value and a text-element sequence consisting of structural elements placed on optimal positions between the words.

### 4.2 Training

For our training we need three different data sets: A training set from which we derive the language model, a cross-validation set on which we adapt the model, and a development testset that is used to stop the training at the right point to prevent over-adaptation. The training set consists of words and structural elements. The other two sets are pure word sequences. Finally we calculate the perplexity of an independent testset that also contains words only. Figure 2 shows the algorithm for the procedure of iterative training. It is designed that two or more structural elements can be inserted in a row. The training set is considered as a closed chain; this means that after the last word has been reached the algorithm proceeds to the first pair of text elements again.

The output are the training set, the cross-validation set, and the development testset, all of them with structural elements placed at optimal positions. Note that there were no changes made on the word order. This distribution of structural elements locally minimises the perplexity of the training corpus. The result and the number of iterations necessary depend on the initial positions of the structuring elements in the training corpus.
Beginning with an initial distribution of structural elements in the training corpus a language model is created.

The perplexity of the cross-validation corpus and the development testset are calculated as explained in section 4.1.

Start at the beginning of the training set.

| Proceed to the next pair of text elements in the training set. |
| Repeat this for all structural elements. |
| Is there already a structural element inserted between the two text elements in the training set? |
| yes | no |
| The structural element is deleted or replaced by another structural element. | A structural element is inserted. |
| Re-calculate the language model. |
| Calculate the perplexity on the cross-validation set. |
| Did the cross-validation perplexity decrease by this operation? |
| yes | no |
| The insertion or deletion is kept in the training corpus. | The insertion or deletion is undone. |
| Calculate perplexity of development testset. |
| Repeat until the perplexity of the development testset has reached a minimum. |

Figure 2: Training algorithm for a structural element language model

5 EXPERIMENTS

For our experiments we used a variant of the algorithm in Figure 2 that processed bigrams and could only insert one structural element between two words. We included the vocabulary of the whole corpus in the language model. That means that some of the 6703 words of the language model vocabulary occur neither during training nor during the test. We used transcriptions of spontaneous speech dialogues for training and testing. Dialogue material is naturally segmented into turns uttered by two different speakers (Weilhammer, Oppermann, and Burger, 2000). Therefore we calculated the perplexity in portions of turns.

The training stops when the perplexity of the development testset reaches its minimum. At this point we have the perplexity and the optimal sequence of structural elements and words for the cross-validation set and the development testset. From this data we built five models using different combinations of data sets and calculated the testset perplexity.

- Only the training set at the point of interruption (train)
- The training set, the cross-validation set and the development testset. Each of them at the point of interruption (test-dc)
- The training set at the point of interruption; the cross-validation set and the development testset with no structural elements inserted (test-odoc)
- The training set and the cross-validation set at the point of interruption; the development testset with no structural elements inserted (test-odc)
The training set and the development test set at the point of interruption; the cross-validation set with no structural elements inserted (test-ocd)

Three different experiments were performed. We used different starting configurations, varied the number of different structural elements to be inserted and compared our model with some standard language models.

### 5.1 The Data for Training and Testing

For training and testing we used dialogues which were recorded and transliterated during the first phase of the Verbmobil project (Wahlster, 2000). In all the recordings two German native speakers were asked to fix a date for a business meeting (Kohler, Lex, Pätzold, Scheffers, Simpson, and Thon, 1994) by using previously prepared diaries. For these recordings transcriptions of orthography and concomitant phenomena were carefully prepared by humans (Burger, 1997). We only used the orthographic transcriptions for our experiments.

<table>
<thead>
<tr>
<th>Vocabulary</th>
<th>Words</th>
<th>Turns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training Set</td>
<td>1358</td>
<td>14161</td>
</tr>
<tr>
<td>Cross-Validation Set</td>
<td>1760</td>
<td>27009</td>
</tr>
<tr>
<td>Development Testset</td>
<td>1905</td>
<td>27621</td>
</tr>
<tr>
<td>Testset</td>
<td>1444</td>
<td>13964</td>
</tr>
</tbody>
</table>

Table 1: Data used for training and tests.

In the tests we want to show that our method works and that the described effects will be found. We selected four parts of the corpus, a small training set, a cross-validation set of almost double the size, a slightly larger development testset and a testset of about the same size as the training set (see Table 1).

### 5.2 Different Starting Configurations

We applied the training algorithm (see Figure 2) on four different initial variants of the training set and carried out a training for both the back-off model and the rest-element model. The parameter $D$ was varied to get an optimal model. In all instances 10 different structural elements were used.

No structural elements inserted (none):

**ja gut Sie haben recht ich hab’ hier meinen Kalender verdreht achtundzwanzigster neunundzwanzigster sieht bei mir auch sehr gut aus sind Sie einverstanden Flugzeug ja**

Inventory: $\langle s_1 \rangle$, $\langle s_2 \rangle$, ..., $\langle s_{10} \rangle$

Structural elements inserted after the 10 most frequent words (most frequent):

**ja $\langle s_{ja} \rangle$ gut Sie haben recht ich $\langle s_{ich} \rangle$ hab’ hier meinen Kalender verdreht achtundzwanzigster neunundzwanzigster sieht bei mir auch sehr gut aus sind Sie einverstanden Flugzeug ja**

Inventory: $\langle s_{ich} \rangle$, $\langle s_{ja} \rangle$, $\langle s_{das} \rangle$, $\langle s_{wir} \rangle$, $\langle s_{dann} \rangle$, $\langle s_{da} \rangle$, $\langle s_{und} \rangle$, $\langle s_{es} \rangle$, $\langle s_{ist} \rangle$, $\langle s_{ih} \rangle$
Figure 3: Back-off model: The training was carried out for 10 different structural elements. Four different starting configurations of the training set were used: No structural elements inserted (none), structural elements inserted after the 10 most frequent words (most frequent), structural elements inserted after each word (every) and structural elements inserted at syntactic and prosodic boundaries (syntactic). The top four diagrams show the perplexities of the cross-validation set (crossval), the development testset (devtest) and the testset (test) after training with the back-off model plotted against the discount value $D$. The testset perplexities of another four extended models are displayed as well. They additionally incorporate the bigrams of the development testset (d) and the cross-validation set (c) with no structural element inserted (o) or the structural element distribution at the point where the training was interrupted. The bottom left plot shows the different testset perplexities that were obtained when the training was interrupted at the minimum perplexity of the development testset. The numbers of training-set bigrams (iterations) that were processed until the training was interrupted are displayed in the bottom right plot.
Figure 4: Rest-element model: The training was carried out for 10 different structural elements. Four different starting configurations of the training set were used: No structural elements inserted (none), structural elements inserted after the 10 most frequent words (most frequent), structural elements inserted after each word (every) and structural elements inserted at syntactic and prosodic boundaries (syntactic). The top four diagrams show the perplexities of the cross-validation set (crossval), the development testset (devtest) and the testset (test) after training with the rest-element model plotted against the discount value $D$. The testset perplexities of another four extended models are displayed as well. They additionally incorporate the bigrams of the development testset (d) and the cross-validation set (c) with no structural element inserted (o) or the structural element distribution at the point where the training was interrupted. The bottom left plot shows the different testset perplexities that were obtained when the training was interrupted at the minimum perplexity of the development testset. The numbers of training-set bigrams (iterations) that were processed until the training was interrupted are displayed in the bottom right plot.
Structural elements inserted after each word (each):
ADP: ja $s_1$ gut $s_2$ Sie $s_3$ haben $s_4$ recht $s_5$ ich $s_6$ hab’ $s_7$ hier $s_8$ meinen $s_9$ Kalender $s_{10}$ verdreht $s_{11}$ achttundzwanzigster $s_{12}$ neunundzwanzigster $s_{13}$ ...
Inventory: $s_1$, $s_2$, ..., $s_{10}$

Structural elements inserted at syntactic and prosodic boundaries (syntactic):
ADP: ja $s_{\text{Comma}}$ gut $s_{\text{Comma}}$ Sie haben recht $s_{\text{Comma}}$ ich hab’ hier meinen Kalender verdreht $s_{\text{fullStop}}$ achttundzwanzigster $s_{\text{Comma}}$ neunundzwanzigster sieht bei mir auch sehr gut aus $s_{\text{fullStop}}$ sind Sie einverstanden $s_{\text{Comma}}$ Flugzeug $s_{\text{Comma}}$ ja $s_{\text{fullStop}}$
Inventory: $s_{\text{uh}}$, $s_{\text{ahm}}$, $s_{\text{hm}}$, $s_{\text{QuestionMark}}$, $s_{\text{CorrectionStart}}$, $s_{\text{CorrectionEnd}}$, $s_{\text{Comma}}$, $s_{\text{fullStop}}$, $s_{\text{RepetitionStart}}$, $s_{\text{RepetitionEnd}}$

In the case of syntactical and prosodic boundaries it was ensured that only one structural element was placed between two words. Since hesitations and filled pauses like ähm, äh and hm are included in the vocabulary the structural elements corresponding to them were placed after the respective items.

5.2.1 Back-Off Model with Different Starting Configurations

The results for the back-off model are displayed in Figure 3. In all four instances the best values for the development-testset perplexity (devtest) are obtained at $D = 0.1$. The testset perplexity of the model created only from the training set (test) has in all cases a minimum at this position too. Only for syntactic boundaries is this just a relative minimum. The absolute minimum for this initial condition is at $D = 0.17$.

The language model, which was built from the training set at the point of interruption plus the development testset and the cross-validation set without structural elements inserted (test-odc) had the best perplexity for almost all values of $D$. It has a minimum around $D = 0.17$. Only in one case was it matched by the model which was built from training set and cross-validation set at the point of interruption plus the development testset in its initial state (test-odc). That is where training started from a word sequence with initially no structural elements inserted. Both reached a perplexity of 379 for $D = 0.17$. By the way, this is the best perplexity obtained for back-off models in this experiment. The model whose training was based on the syntactical positions ended up with the poorest performance.

The models differ in the iterations needed for training (see bottom right plot). In this respect the models whose training was based on the plain word chain (none) resulted in the best performance, followed by the models obtained from a training set with structural elements placed after the most frequent words.

5.2.2 Rest-Element Model with Different Starting Configurations

The results for the rest-element model are displayed in Figure 4. The best values for the development-testset perplexity (devtest) are obtained at $D = 0.3$ and $D = 0.6$. These are also minima for the testset perplexity of the model created only from the training set (test). The language model which was built from the training set and the cross-validation set at the point of interruption plus the development testset in its initial state (test-odc) had the best perplexity for all values of $D$ and all initial settings. But its minimum usually occurred at a higher discount value $D$ than the perplexity minimum of the development testset.
The test-odc model built from a training set with initially no structural elements inserted yields the best perplexity of 277 at $D = 0.3$ for the devtest-minimum criterion. The lowest value of 268 would have been reached at $D = 0.6$. At this point the development testset perplexity has a local minimum as well. Again the training with the syntactical model ended up with the poorest performance.

Processing time varies from 2 to 28 iterations over the training set. The bottom right plot in Figure 4 shows the number of bigrams that were processed during the training. The models obtained from the training set with structural elements placed after the most frequent words show the best convergence. They need the lowest number of iterations. Training, that was started from a word sequence with structural elements inserted after every word, results in the highest numbers of iterations.

At the values $D_{\text{min}}$ where devtest-perplexity reaches its minimum the number of iterations also has a local minimum.

5.3 Different Numbers of Structural Elements

We used the training set with initially no structural elements inserted and carried out a training for 1, 10, 50 and 100 different structural elements. Please note that the 10-structural-element models in this experiment are identical with the model that was trained with the text-element sequence that initially consisted of pure words in section 5.2.

5.3.1 Back-Off Model with Different Numbers of Structural Elements

The results for the back-off model are displayed in Figure 5. The best values for the development-testset perplexities are obtained at $D = 0.1$. In most cases they are close by ($E = 50, 100$) or even at the minimum ($E = 1, 10$) of the testset perplexity of the model created only from the training set (test).

The language model, built from the training set at the point of interruption plus the development testset and the cross-validation set without structural elements inserted (test-odc) performs very well in all cases. It often has a minimum in the region of $0.13 \leq D \leq 0.20$. For small numbers of structural elements ($E = 1, 10$) the model which was constructed from training set and development testset at the point of interruption plus the cross-validation set without structural elements inserted (test-odc) performed better.

With the devtest-minimum criterion a minimum perplexity of 363 was reached for the test-odc model and one structural element ($D = 0.1$). In general the performance of the language models became poorer as the number of structural elements that could be inserted increased.

The bottom right plot in Figure 5 shows the number of bigrams that were processed during the training. Please note that the processing time for an iteration depends on the number of structural elements that have to be inserted. The same number of structural elements has to be tested for each bigram of the cross-validation set. Therefore the time for each iteration increases proportionally to the number of structural elements squared.

In the bottom right plot the number of iterations processed until the devtest-minimum is reached remains almost constant at a low level for all values of $D$ in the 1-structural-element model. At small values of $D$ this is true for the other models as well, but for higher values of $D$ they consume more iterations.
Figure 5: Back-off model: The training was carried out for 1, 10, 50 and 100 different structural elements. In the initial training set there were no structural elements inserted. The top four diagrams show the perplexities of the cross-validation set (crossval), the development testset (devtest) and the testset (test) after training with the back-off model plotted against the discount value $D$. The testset perplexities of another four extended models are displayed as well. They additionally incorporate the bigrams of the development testset (d) and the cross-validation set (c) with no structural element inserted (o) or the structural element distribution at the point where the training was interrupted. The bottom left plot shows the different testset perplexities that were obtained when the training was interrupted at the minimum perplexity of the development testset. The numbers of training-set bigrams (iterations) that were processed until the training was interrupted are displayed in the bottom right plot.
Figure 6: Rest-element model: The training was carried out for 1, 10, 50 and 100 different structural elements. In the initial training set there were no structural elements inserted. The top four diagrams show the perplexities of the cross-validation set (crossval), the development testset (devtest) and the testset (test) after training with the rest-element model plotted against the discount value $D$. The test-set perplexities of another four extended models are displayed as well. They additionally incorporate the bigrams of the development testset (d) and the cross-validation set (c) with no structural element inserted (o) or the structural element distribution at the point where the training was interrupted. The bottom left plot shows the different testset perplexities that were obtained when the training was interrupted at the minimum perplexity of the development testset. The numbers of training-set bigrams (iterations) that were processed until the training was interrupted are displayed in the bottom right plot.
5.3.2 Rest-Element Model with Different Numbers of Structural Elements

The results for the rest-element model are displayed in Figure 6. The best values for the development-testset perplexity are obtained at $D = 0.3$ (10, 50, 100 structural elements) and $D = 0.4$ (1 structural element). The absolute minima of the testset perplexity of the models created only from the training set (test) are reached at $D = 0.3$ for 1, 10, 50 structural elements. For 100 structural elements this is the case at $D = 0.5$ (There is a local minimum at $D = 0.3$).

As in the experiment before, the language model which was built from the training set and the cross-validation set at the point of interruption plus the development testset in its initial state (test-odc) had the best perplexity for all values of $D$ and its minimum usually occurs at a higher discount value than the perplexity minimum of the development testset.

The test-odc model with only one structural element yields the best perplexity of 260 at $D = 0.3$ for the devtest-minimum criterion. The lowest value of 254 would have been reached at $D = 0.7$. In general the performance of the language models became poorer as the number of structural elements that could be inserted increased.

The bottom right plot in Figure 6 shows the number of bigrams that were processed during the training. As explained above the processing time for an iteration depends quadratically on the number of structural elements $E$.

The 10-structural-element models are the slowest in terms of iterations. For $0.1 \leq D \leq 0.6$ the 100-structural-element models are the fastest although their iterations are by far the most time consuming in this experiment. For higher values of $D$ the number of iterations increases. The 50-structural-element models are slightly worse. The best performers in this experiment are the 1-structural-element models, which consume around 30000 iterations for all values of $D$.

At the values $D_{\text{min}}$ where the devtest-perplexity reaches its minimum, the number of iterations of all rest-element models in this experiment are moderate. Interestingly, the graph for the iterations of the 1-structural element model has a minimum at $D = 0.7$, that is identical to the position of the minimum-test-odc-perplexity.

5.4 Comparison with Standard Models

Finally we will compare the best results of the back-off model and the rest-element model obtained in the previous experiments with those of well established techniques such as Good Turing discounting (Katz, 1987) and a class-based bigram model as described in Brown, Pietra, deSouza, Lai, and Mercer (1992) and in Kneser and Ney (1993).

We used two well known language model tool-kits to build our reference models. With the CMU-Cambridge Language Model Tool-Kit (CMU-LM) by Clarkson and Rosenfeld (1997) (see also Rosenfeld, 1994) we constructed the Good Turing model. The class-based model was prepared with the SRI Language Model Tool-Kit (SRI-LM) by Stolke. We built models for 300, 400 and 500 classes and took the 400-classes-model which had the best perplexity on the training set.

We combined the three distinct data sets that were previously referred to as the training set, the cross-validation set and the development testset for the training of both reference models, and calculated the perplexity of the testset (see Table 1). The results are summarised in Table 2.
<table>
<thead>
<tr>
<th>Model</th>
<th>Perplexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Structural Element (back-off)</td>
<td>363</td>
</tr>
<tr>
<td>Structural Element (rest element)</td>
<td>260</td>
</tr>
<tr>
<td>Good Turning Discounting</td>
<td>212</td>
</tr>
<tr>
<td>Class-Based Bigram (400 classes)</td>
<td>135</td>
</tr>
</tbody>
</table>

Table 2: Perplexities of two reference models and both back-off and rest-element model with 1 structural element trained from a pure word sequence.

CONCLUSION

In all experiments the rest-element model works better than the back-off model. The best model is obtained with 1 structural element and an initial training set that consists of words only and has no structural elements inserted. This indicates that the number of parameters that are estimated during the training is high compared to the size of the data. For a robust computation of the probability values larger training and cross-validation sets must be used. The fact that the class-based model, which has the lowest number of free parameters, performs best supports this hypothesis.

Since the structural element models do not attain the results of standard methods we will present some suggestions for improvement.

Comparison of the results of the back-off model with the results of the rest-element model shows the influence of the probability distribution on the perplexity. It might be interesting to combine the rest-element model with Good Turing or Witten Bell discounting.

A different approach is to improve the training algorithm, since the best results were obtained for the 1-structural-element model on an initially plain training set. It might be interesting to take the training set at the point of interruption and start a new training for a 2-structural-element model, which might have a better performance.

Another strategy is inspired from an analysis of the four top plots in Figure 6. They show an interesting phenomenon. The testset perplexity together with the development-testset perplexity increases as the perplexity of the cross-validation set decreases. It might be interesting to use this for a new training algorithm that, instead of minimising the cross-validation perplexity and stopping at the minimum development-testset perplexity, minimises the difference between these two measures.

The main problem of the training algorithm is that the training set, the cross-validation set and the development testset are not fully exploited. The first step to improving this is to find an optimal division of the data into a training set, a cross-validation set and a development testset. In a second step it would be interesting to implement a leave-one-out training for structural elements.

Although the results obtained with the structural element approach could not match those of standard models, we think that our results are promising. We are especially encouraged by the fact that this method and especially the training algorithm is only at an early stage of its development. We would point out that no previous knowledge is needed for the unsupervised training of the best model. Furthermore, the concept of structural elements being placed within word sequences is interesting from a theoretical point of view, because it suggests a new view of the problem of structuring text and provides an appropriate formalism.
References


