ABSTRACT
In this paper, we propose a new method for computing and applying language model look-ahead in a dynamic network decoder, exploiting the sparseness of backing-off n-gram language models. Only partial (sparse) look-ahead tables are computed, with a size that depends on the number of words that have an n-gram score in the language model for a specific context, rather than a constant, vocabulary dependent size. Since high order backing-off language models are inherently sparse, this mechanism reduces the runtime- and memory effort of computing the look-ahead tables by magnitudes. A modified decoding algorithm is required to apply these sparse LM look-ahead tables efficiently. We show that sparse LM look-ahead is much more efficient than the classical method, and that full n-gram look-ahead becomes favorable over lower order look-ahead even when many distinct LM contexts appear during decoding.

Index Terms— speech, recognition, decoding, search, language model, look-ahead

1. INTRODUCTION
In a dynamic network decoder, the language model (LM) and acoustic model (AM) are combined dynamically. The acoustic model is used to build a compact HMM search network representing all the words in the vocabulary, with word-end labels representing the words in the vocabulary attached to specific states of the network. During decoding state hypotheses are propagated through the search network to find the most probable path, and pruning is used to limit the search to promising state hypotheses. LM probabilities are applied to hypotheses as soon as word-end labels are reached.

The idea of LM look-ahead is to integrate the LM knowledge into the pruning process already before reaching the word-end labels, potentially leading to a faster and more precise decoding [1]. From the HMM search network, a compressed look-ahead network is generated, generally referred to as the look-ahead tree, consisting of look-ahead nodes. For each LM context that appears during the decoding, a look-ahead table needs to be computed that assigns a probability to each look-ahead node.

If many different LM contexts appear during decoding, the computation of the look-ahead tables becomes a significant efficiency problem. To overcome this problem it is common to use lower-order LM look-ahead. When reducing the order of the LM look-ahead to unigram, only unigram LM probabilities are distributed over the search network, and only one single look-ahead table needs to be computed. Using bigram look-ahead, one look-ahead table needs to be computed for each possible predecessor word that appears during search.

In [2], the computation of look-ahead tables is sped up by using lower order \((n-1)\)-gram look-ahead tables as initialization for higher order \(n\)-gram tables, and only updating those parts of the table which are not estimated through the LM back-off in the given context. This strategy can potentially reduce the effort of computing the table, however there is still one full look-ahead table that needs to be allocated and copied for each \(n\)-gram context, and the authors of [3] even state that the efficiency is reduced when using a 4-gram LM and a large vocabulary, since many additional back-off tables need to be computed.

In this paper we propose a method that computes a sparse look-ahead table for each LM context, which only assigns probabilities to those look-ahead nodes that are not estimated through the LM back-off in the given context.

Sparse LM look-ahead tables only assign probabilities to a subset of all look-ahead nodes, so we need to use lower order tables as back-off as soon as the decoder reaches a node which is not covered in the corresponding sparse table, however, unlike [2], we don’t need to copy the back-off tables.

Previous methods treat the LM look-ahead as a kind of black-box separated from the decoder. With our method, we directly integrate the decoder, the LM look-ahead and the structure of the \(n\)-gram LM, in order to improve the efficiency.

Our decoder is based on the word conditioned tree search [4] (WCS) concept (which is more or less equivalent to the token passing [5] concept).

2. WORD CONDITIONED TREE SEARCH
In WCS, one virtual copy of the search network is created for each encountered LM context \(h\) (for simplicity reasons we will call it a tree copy even though this term does not apply in case of using phonetic contexts across word boundaries). When two partial paths cross in one tree copy, they are recombined, i.e. only the better path needs to be kept.

A state hypothesis is denoted by the tuple \((s, h)\), where \(s\) is an HMM state, and \(h = u_1^{m-1}\) is an LM context (the \(m-1\) predecessor words). \(S(t)\) denotes the set of state hypotheses that are active at time frame \(t\).

\(Q_h(t, s)\) denotes the probability of the best path through the HMM network that ends at timeframe \(t\) in state \(s\) with LM context \(h\), and is defined for all active state hypotheses \((s, h) \in S(t)\).
Acoustic pruning is used to reduce the number of active state hypotheses, by applying a beam and removing all state hypotheses that have a probability lower than the best one multiplied by a threshold \( f_{AC} \) (when providing pruning threshold values, we will use the negative logarithms \(-\log(f_{AC})\), as those values are more descriptive).

As soon as a word end state \( S_w \) is reached, a word end hypothesis is created, where the LM probability \( p(w|h) \) in the corresponding context \( h \) is applied to the probability.

LM pruning is used to reduce the number of word end hypotheses, by applying a further pruning threshold \( f_{LM} \) to all word end hypotheses of a timeframe. The word end hypotheses that survive the LM pruning are used to activate the root states of following tree copies.

3. STANDARD LM LOOK-AHEAD

Let \( Q_{max}(t) \) be the probability of the overall best state hypothesis of timeframe \( t \):

\[
Q_{max}(t) = \max_{(s, h) \in S(t)} Q_h(t, s) \tag{1}
\]

During acoustic pruning, only those state hypotheses are preserved that have a probability higher than \( Q_{max}(t) \cdot f_{AC} \):

\[
S_{AC}(t) = \{(s, h) | (s, h) \in S(t) \land Q_h(t, s) > Q_{max}(t) \cdot f_{AC}\} \tag{2}
\]

Where \( f_{AC} < 1 \) is the acoustic pruning threshold.

When LM look-ahead is used, the knowledge about reachable word ends is included during the acoustic pruning. Let \( W(s) \) be the set of word ends that are reachable from state \( s \). The exact LM look-ahead probability of the best word that can be reached from state \( s \) for history \( h \) is then defined as:

\[
p_h(s) := \max_{w \in W(s)} p(w|h) \tag{3}
\]

The state hypothesis probability extended by the LM look-ahead probability is:

\[
\bar{Q}_h(t, s) = Q_h(t, s) \cdot p_h(s) \tag{4}
\]

And the acoustic pruning, as per Equation (1) and (2), is then performed on \( \bar{Q}_h(t, s) \) instead of \( Q_h(t, s) \).

In order to supply the decoder with the look-ahead probabilities \( \pi_h(s) \) efficiently, a compressed version of the search network is generated [1]. A look-ahead tree is constructed so that all states \( s \) that have an equal set of reachable word ends \( W(s) \) share the same look-ahead node \( n \), and exactly the words \( w \in W(s) \) are reachable as successor leafs from node \( n \). The mapping \( n = l(s) \) maps from each state to its corresponding look-ahead node. As a kind of approximation, the look-ahead tree can be truncated at a specific depth \( h \) w.r.t. the phoneme position within a word, to reduce the number of look-ahead nodes \( N \). Typically the number of nodes depends on the vocabulary size \( V \), and ranges between \( V/3 \) and \( V \cdot 2 \), depending on how much the tree is truncated.

A look-ahead table for context \( h \) assigns a probability to each look-ahead node \( n \):

\[
b_h(n) = \max_{w \in W(n)} p(w|h) \tag{5}
\]

Where \( W(n) \) are the word ends reachable from look-ahead node \( n \). During decoding, the look-ahead table is then used to assign look-ahead probabilities to state hypotheses, by mapping each state \( s \) to its look-ahead node \( l(s) \), and reading the probability from the look-ahead table:

\[
\bar{Q}_h(t, s) = Q_h(t, s) \cdot b_h(l(s)) \tag{6}
\]

Typically, the whole look-ahead table \( b \) is filled during decoding as soon as the context \( h \) is encountered. The probabilities can be computed efficiently by sorting the look-ahead nodes topologically, filling the leaf nodes with the word-probabilities from the LM, and then propagating the probabilities in reverse-topological order. The overall effort is linear in \( N \) for each new context \( h \) that is encountered.

4. N-GRAM LANGUAGE MODELS

An n-gram LM uses the sequence \( h \) of the \( n-1 \) predecessor words to predict the probability of the successor word \( w \):

\[
p(w|h) := \begin{cases} f(w|h) & \text{if } C(h, w) > 0 \\ p(w|h') \cdot o(h) & \text{else (use back-off)} \end{cases} \tag{7}
\]

Where \( C(h, w) \) is the count how often the word \( w \) has been seen in the context of the \( n-1 \) word history \( h \) while training the LM, \( o(h) \) is the back-off parameter of history \( h' \) is the lower order history of \( h \) (the \( n-2 \) predecessor words), and \( f(w|h) \) is the discounted probability stored in the LM.

When using high-order language models like for example 4-grams, even with huge amounts of training data, most of the probabilities \( p(w|h) \) are estimated through backing-off, because the number of possible 4-grams is \( V^4 \), which can hardly be covered all, when the number \( V \) of words in the vocabulary is large.

In a backing-off LM, the set \( B(h) = \{w|C(h, w) > 0\} \) of words that have an \( n \)-gram in a specific context \( h \), together with the according probabilities \( f(w|h) \), can be accessed very efficiently.

5. SPARSE LM LOOK-AHEAD

The idea of the sparse LM look-ahead method introduced here is to only compute look-ahead table entries \( b_h(n) \) for nodes \( n \) that lead to at least one word \( w \in W(n) \) with a full \( n \)-gram in the LM (e.g. \( w \in B(h) \)). We completely ignore all other nodes, without even allocating memory for them. On average, the number of affected nodes is very low in comparison to the overall number of nodes. We thereby disconnect the effort required for LM look-ahead from the size of the complete look-ahead table \( N \) (and thus from the size of the vocabulary \( V \)).

5.1. Approximation

For a given history \( h \), presume that the full \( n \)-gram probabilities all are larger than the probabilities from backing-off:

\[
\min_{w \in B(h)} p(w|h) > \max_{w \notin B(h)} p(w|h) \tag{8}
\]

Then we can compute the probabilities of all look-ahead nodes \( n \) with \( W(n) \cap B(h) \neq \emptyset \) by only considering the words \( w \in B(h) \), and completely ignoring all the words with probabilities that are estimated through backing-off (see Equation (5)). Depending on the discounting method, Equation (8) does not necessarily hold. However, with the popular discounting methods like modified Kneser-Ney discounting, the event that the equation is negated is quite unlikely.

As an approximation we will assume Equation (8) to hold for our LM look-ahead method. We have experimented with ways to avoid this approximation, however the approximation does not lead to any measurable problems, while allowing a more efficient calculation of the sparse LM look-ahead tables.
5.2. Sparse Look-Ahead Tables

The sparse LM look-ahead tables are implemented using fixed-size hash tables, with a predicted table size $s(h) = fac \cdot nc(h)$, where $nc(h)$ assigns the expected number of look-ahead nodes that result from a specific number of non-back-off probabilities $|B(h)|$, and $fac$ is a factor that decides how many clashes will happen in the hash table. The mapping $nc(h)$ is learned online during decoding using a simple linear prediction. We use $fac = 1.8$, which leads to only about one clash in 4 hash table entries.

Each entry in a sparse LM look-ahead table consists of 4 bytes for the actual probability, plus 4 bytes of administrative overhead for the hash table. A non-sparse LM look-ahead table consists of 4 bytes for each of the $N$ nodes, thus a sparse table is only more compact than the full table if $2 \cdot fac \cdot nc(h) < N$. This condition is satisfied in most cases, especially for large vocabularies.

We can fill a sparse LM look-ahead table efficiently using the following algorithm:

**Preparation:**
- For each node $n$ assign a topological depth $d(n)$ and remember a list $\text{parent}(n)$ of parent nodes.
- For each word $w$ remember the leaf node $n_w$ in which the word appears.

**Data structures:**
- For each depth $depth$, manage a queue of waiting nodes $\text{queue}(depth) = \{(n_1, q_1), \ldots\}$, with nodes $n_i$ and probabilities $q_i$.

**Initialization:**
- For each word $w \in B(h)$, add $(n_w, f(w|h))$ into $\text{queue}(d(n_w))$.

**Iteration:**

```plaintext
foreach depth in reverse-topological order do
    foreach n with $(n, \cdot) \in \text{queue}(depth)$ do
        Compute maximum:
        $prob(n) \leftarrow \max_{q(n,q)\in \text{queue}(depth)} q$.
        Insert into table: $b_h(n) \leftarrow prob(n)$.
        foreach par in parent(n) do
            Add $(par, prob(n))$ into $\text{queue}(d(par))$.
        end
    end
end
```

When the algorithm has finished, all look-ahead table entries $b_h(n)$ for nodes $n$ with $W(n) \cap B(h) \neq \emptyset$ are filled.

It is important to note that, on each topological level $depth$, the maximum probability $prob(n)$ can be computed using a simple pre-allocated global array of size $N$ (which is more efficient than using the hash table). There are no complex data structures involved, and the overall effort is $O(nc(h))$.

There is a specific node-count $nc(h)$ from which a non-sparse LM look-ahead table becomes more efficient to compute than a sparse table, and thus we use sparse LM look-ahead tables below that threshold only. We only create sparse LM look-ahead tables if $nc(h) < \text{thres} \cdot N$. We use a threshold of $\text{thres} = 0.5$, which leads to a very good efficiency for all our test cases (generally it seems better to pick a too high threshold than a too low threshold).

5.3. Decoding

The decoder requires the look-ahead table entries during acoustic pruning at timeframe $t$ for state hypothesis $(s, h)$ to compute the look-ahead probability $Q_h(t, s)$ (see Equation (6)). When the decoder reaches a state hypothesis $(s, h)$ for which the look-ahead node $n = h(s)$ is not filled in the sparse LM look-ahead table $b_h$, it means that the probabilities of all word ends $W(s)$ will be estimated through the back-off (see Equation (7)).

We could apply the back-off parameter $o(h)$ to the probabilities from the lower order look-ahead $b_{h'}$ to calculate the correct probability: $Q_h(t, s) = Q_h(t, s) \cdot o(h) \cdot b_{h'}(n)$ (given that $b_{h'}$ is defined). For efficiency reasons, it is not desirable to eventually follow the cascade from $h$ to $h'$ again and again at each timeframe whenever a look-ahead probability $Q_h(t, s)$ is required by the decoder.

Instead, we assign a look-ahead history $h_t$ to each virtual tree copy (and thus to each state hypothesis), and whenever we encounter a state hypothesis $(s, h_t)$ where the look-ahead table entry $b_h((s))$ is not defined, we apply the back-off parameter $o(h_t)$ directly to the state hypothesis probability, and transform the state hypothesis into $(s, h, h_t')$ so that it depends on the lower order look-ahead history $h_t'$. Thus, we apply the back-off parameter $o(h)$ only when crossing from a look-ahead node $n$ with a defined probability $b_{h_t}(n)$ into a look-ahead node $n'$ with undefined probability $b_{h_t'}(n')$.

6. EXPERIMENTAL RESULTS

We perform all experiments on a modified variant of the RWTH Aachen open source speech recognition software [6]. We use a speaker-independent recognition system based on the Quaero 2008 German evaluation system, with a 200k word vocabulary and 270k pronunciations, and an acoustic model with 4501 gaussian mixture models consisting of 551450 Gaussian densities using a globally tied diagonal covariance matrix. The 4-gram LM consists of 10 million 4-grams, 13 million 3-grams, 31 million bigrams, and 200k unigrams. We perform all experiments on the first pass of our multi-pass system, without speaker adaptation. Overall, the system was tuned to reach the best possible word error rate (WER), without any consideration of the efficiency. As test-corpus, we use 61 minutes of speech recorded from the SWR TV broadcast news. The real time factors (RTF) are calculated on a 2.6 GHz Intel Core2 Duo machine with 4 GB of memory.

The overall time spent with LM look-ahead table generation directly depends on the number of distinct LM contexts that appear during decoding. That number depends on the involved models, and also on the LM pruning threshold $f_{LM}$ (see Section 2). For our experiments, we have chosen the lowest possible LM pruning threshold that still allows reaching the best possible WER, and then we have left this pruning threshold fixed.

Using the acoustic pruning threshold 250 (which leads to the best possible WER), at each timeframe $t$ we encounter about 1.66 unique 3-word contexts $h$ for which we need to calculate new look-ahead tables. The unpruned look-ahead tree has 53k look-ahead nodes $N$.

Table 1 shows the RTF and the number of state hypotheses after pruning, under varied acoustic pruning $fac$ (see Section 2). The table shows that, in general, a higher LM look-ahead order leads to less state hypotheses and a better WER at the
Table 1. WER, RTF and number of state hypotheses after pruning in combination with different LM look-ahead methods, under varied acoustic pruning.

<table>
<thead>
<tr>
<th>LA Order</th>
<th>Pruning</th>
<th>150</th>
<th>180</th>
<th>200</th>
<th>250</th>
<th>320</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unigram</td>
<td>WER</td>
<td>13.7%</td>
<td>10.8%</td>
<td>10.4%</td>
<td>10.0%</td>
<td>10.0%</td>
</tr>
<tr>
<td></td>
<td>States</td>
<td>1.2k</td>
<td>1.4k</td>
<td>1.8k</td>
<td>2.0k</td>
<td>2.4k</td>
</tr>
<tr>
<td></td>
<td>RTF</td>
<td>0.89</td>
<td>1.26</td>
<td>1.36</td>
<td>1.96</td>
<td>2.37</td>
</tr>
<tr>
<td>Bigram</td>
<td>WER</td>
<td>12.0%</td>
<td>10.2%</td>
<td>10.0%</td>
<td>9.9%</td>
<td>9.8%</td>
</tr>
<tr>
<td></td>
<td>States</td>
<td>0.86k</td>
<td>1.25k</td>
<td>1.85k</td>
<td>1.95k</td>
<td>2.65k</td>
</tr>
<tr>
<td></td>
<td>RTF</td>
<td>2.35</td>
<td>3.65</td>
<td>4.12</td>
<td>4.99</td>
<td>6.65</td>
</tr>
<tr>
<td>Sparse</td>
<td>WER</td>
<td>11.7%</td>
<td>10.2%</td>
<td>9.9%</td>
<td>9.9%</td>
<td>9.8%</td>
</tr>
<tr>
<td></td>
<td>States</td>
<td>0.74k</td>
<td>1.22k</td>
<td>1.62k</td>
<td>2.23k</td>
<td>2.73k</td>
</tr>
<tr>
<td></td>
<td>RTF</td>
<td>3.07</td>
<td>4.15</td>
<td>4.64</td>
<td>5.95</td>
<td>7.65</td>
</tr>
<tr>
<td>3-gram</td>
<td>WER</td>
<td>11.8%</td>
<td>10.3%</td>
<td>10.0%</td>
<td>9.8%</td>
<td>9.8%</td>
</tr>
<tr>
<td></td>
<td>States</td>
<td>0.72k</td>
<td>2.1k</td>
<td>1.41k</td>
<td>2.16k</td>
<td>2.96k</td>
</tr>
<tr>
<td></td>
<td>RTF</td>
<td>2.95</td>
<td>4.02</td>
<td>4.51</td>
<td>5.75</td>
<td>7.47</td>
</tr>
<tr>
<td>Sparse</td>
<td>WER</td>
<td>1.04</td>
<td>1.13</td>
<td>1.53</td>
<td>2.17</td>
<td>3.47</td>
</tr>
</tbody>
</table>

equal acoustic pruning threshold. To reach a WER close to the optimum of 9.8%, at least bigram LM look-ahead is required. The RTF however rises significantly when non-sparse LM look-ahead is used, due to the costs of the computed look-ahead tables (for example from 1.96 to 4.99 for bigram look-ahead at an acoustic pruning threshold of 250). Sparse LM look-ahead on the other hand only adds a slight overhead over the unigram look-ahead, while still improving the WER. The approximation from Subsection 5.1 does not lead to a measurable effect on the WER.

Figure 1 further illustrates the relationship between RTF and WER of the different look-ahead methods. While the best possible error rate is only reachable with higher order look-ahead, unigram LM look-ahead is still more efficient than standard bigram LM look-ahead for the slightly higher error rates. Sparse LM look-ahead on the other hand is always more efficient than unigram look-ahead, while still allowing to reach the best possible WER. For higher WERs, the efficiency curves of sparse LM look-ahead and unigram look-ahead run very close to each other, however, this is because only the acoustic pruning was varied to generate the graph. When compromising WER, the LM pruning can be reduced as well, which in turn slightly increases the WER, but significantly reduces the number of encountered contexts, and thus the number of look-ahead tables that need to be calculated, as well.

7. CONCLUSIONS AND OUTLOOK

We have shown that sparse LM look-ahead is by magnitudes more efficient than the classical method.

We have achieved similar results (not reported here) across multiple different tasks with different vocabulary sizes (50k english and 600k arabic), although the effect is higher with larger vocabularies.

With a very efficient LM look-ahead, one of the major disadvantages of dynamic decoders in comparison to WFST decoders [7] might be compensated. In future we will compare our decoding architecture thoroughly with the WFST approach.

Since sparse LM look-ahead tables are very small, it would be possible to pre-compute all tables statically, thereby resolving most of the remaining effort for table calculation.

8. ACKNOWLEDGEMENTS

This work was partly realized under the Quaero Programme, funded by OSEO, French State agency for innovation.

9. REFERENCES


