Scalable Modified Kneser-Ney Language Model Estimation

Kenneth Heafield    Ivan Pouzyrevsky    Jonathan H. Clark
Phili"pp Koehn

University of Edinburgh, Carnegie Mellon, Yandex

6 August, 2013
Estimating LMs is Costly

MIT  RAM
SRI  RAM, time
IRST  RAM, time, approximation
Berkeley  RAM, time, approximation
Estimating LMs is Costly

MIT RAM
SRI RAM, time
IRST RAM, time, approximation
Berkeley RAM, time, approximation

Microsoft Delay some computation to query time
Google 100–1500 machines, optional stupid backoff
This Work

- Disk-based streaming and sorting
- User-specified RAM
- Fast
- Interpolated modified Kneser-Ney

7.7% of SRI’s RAM, 14% of SRI’s wall time
Outline

1. Estimation Pipeline
2. Streaming and Sorting
3. Experiments
Text

Counting

Combine and Sort

Adjusting

Sort

Sort

Divide

Interpolate Orders

Sum & Backoff

ARPA/Binary File
Counting

\( \langle s \rangle \) Australia is one of

\[
\begin{array}{|c|c|}
\hline
\text{3-gram} & \text{Count} \\
\hline
\langle s \rangle \text{ Australia is} & 1 \\
\text{Australia is one} & 1 \\
is one of & 1 \\
\hline
\end{array}
\]

Combine in a hash table, spill to merge sort.
Adjusting

Adjusted counts are:

- **Trigrams** Same as counts.
- **Others** Number of unique words to the left.
Adjusting counts are:

**Trigrams**  Same as counts.

**Others**  Number of unique words to the left.

<table>
<thead>
<tr>
<th>Suffix Sorted</th>
<th>Input</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 2 1</td>
<td>are one of</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>is one of</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>are two of</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th><strong>Output</strong></th>
<th><strong>1-gram</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>Adjusted</strong></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>of</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th><strong>Output</strong></th>
<th><strong>2-gram</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>Adjusted</strong></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>one of</td>
<td></td>
</tr>
<tr>
<td></td>
<td>two of</td>
<td>1</td>
</tr>
</tbody>
</table>
Calculating Discounts

Count singletons, doubletons, tripletons, and quadrupletons for each order.

Chen and Goodman

discount$_n$
Text

Counting✓

Adjusting✓

Combine and Sort

Sort

Sum & Backoff

Divide

Sort

Interpolate Orders

Sort

Discounts

ARPA/Binary File
Discounting and Normalization

\[
pseudo(w_n | w_1^{n-1}) = \frac{\text{adjusted}(w_1^n) - \text{discount}_n(\text{adjusted}(w_1^n))}{\sum_x \text{adjusted}(w_1^{n-1}x)}
\]

Save mass for unseen events

Normalize

Estimating

Streaming and Sorting

Experiments
Discounting and Normalization

\[
pseudo(w_n | w_1^{n-1}) = \frac{\text{adjusted}(w_1^n) - \text{discount}_n(\text{adjusted}(w_1^n))}{\sum_x \text{adjusted}(w_1^{n-1} x)}
\]

Save mass for unseen events

Normalize

### Context Sorted Input

<table>
<thead>
<tr>
<th>Context Sorted</th>
<th>Input</th>
<th>Adjusted</th>
</tr>
</thead>
<tbody>
<tr>
<td>are one</td>
<td>of</td>
<td>1</td>
</tr>
<tr>
<td>are one</td>
<td>that</td>
<td>2</td>
</tr>
<tr>
<td>is one</td>
<td>of</td>
<td>5</td>
</tr>
</tbody>
</table>

### Output

<table>
<thead>
<tr>
<th>3-gram</th>
<th>Pseudo</th>
</tr>
</thead>
<tbody>
<tr>
<td>are one of</td>
<td>0.26</td>
</tr>
<tr>
<td>are one that</td>
<td>0.47</td>
</tr>
<tr>
<td>is one of</td>
<td>0.62</td>
</tr>
</tbody>
</table>
Denominator Looks Ahead

\[
\text{pseudo}(w_n | w_1^{n-1}) = \frac{\text{adjusted}(w_1^n) - \text{discount}_n(\text{adjusted}(w_1^n))}{\sum_x \text{adjusted}(w_1^{n-1}x)}
\]

Save mass for unseen events

Normalize

<table>
<thead>
<tr>
<th>Context Sorted Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 1 3 Adjusted</td>
<td>3-gram Pseudo</td>
</tr>
<tr>
<td>are one of that of</td>
<td>are one of</td>
</tr>
<tr>
<td>are one of that of</td>
<td>are one that</td>
</tr>
<tr>
<td>is one</td>
<td>is one of 0.62</td>
</tr>
</tbody>
</table>

Estimating Streaming and Sorting Experiments
Two Threads

<table>
<thead>
<tr>
<th>Sum Thread</th>
<th>Divide Thread</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 1</td>
<td>2 1</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Adjusted</td>
<td>Adjusted</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

Reads ahead and sums

Reads behind to normalize

```
are one of 1
are one that 2
is one of 5
```

```
are one of 1
are one that 2
is one of 5
```

sum = 3
Computing Backoffs

Backoffs are penalties for unseen events.

Bin the entries “are one x” by their adjusted counts

\[ \text{continue}(\text{are one}) = (\text{number with adjusted count 1}, \ldots \text{adjusted count 2}, \ldots \text{adjusted count } \geq 3) \]

Compute backoff in the sum thread

\[ \text{backoff}(\text{are one}) = \text{continue}(\text{are one}) \cdot \text{discount} \]

\[ \sum_{x} \text{adjusted}(\text{are one } x) \]
Computing Backoffs

Backoffs are penalties for unseen events.

Bin the entries “are one x” by their adjusted counts

\[
\text{continue}(\text{are one}) = (\text{number with adjusted count } 1, \ldots \text{adjusted count } 2, \ldots \text{adjusted count } \geq 3)
\]

Compute backoff in the sum thread

\[
\text{backoff}(\text{are one}) = \frac{\text{continue}(\text{are one}) \cdot \text{discount}_3}{\sum_x \text{adjusted}(\text{are one } x)}
\]
Interpolate unigrams with the uniform distribution.

\[ p(\text{of}) = \text{pseudo(\text{of}) + backoff(\epsilon)} \frac{1}{|\text{vocabulary}|} \]
Interpolate Orders

Interpolate unigrams with the uniform distribution,

\[ p(\text{of}) = \text{pseudo}(\text{of}) + \text{backoff}(\epsilon) \frac{1}{|\text{vocabulary}|} \]

Interpolate bigrams with unigrams, etc.

\[ p(\text{of} | \text{one}) = \text{pseudo}(\text{of} | \text{one}) + \text{backoff}(\text{one}) p(\text{of}) \]
Interpolate Orders

Interpolate unigrams with the uniform distribution,

\[ p(\text{of}) = \text{pseudo}(\text{of}) + \text{backoff}(\epsilon) \frac{1}{|\text{vocabulary}|} \]

Interpolate bigrams with unigrams, etc.

\[ p(\text{of} | \text{one}) = \text{pseudo}(\text{of} | \text{one}) + \text{backoff}(\text{one}) p(\text{of}) \]

<table>
<thead>
<tr>
<th>Suffix</th>
<th>Lexicographic Sorted Input</th>
<th>interpolation weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>of</td>
<td>0.1 backoff(\epsilon) = 0.1</td>
<td></td>
</tr>
<tr>
<td>one of</td>
<td>0.2 backoff(one) = 0.3</td>
<td></td>
</tr>
<tr>
<td>are one of</td>
<td>0.4 backoff(are one) = 0.2</td>
<td></td>
</tr>
</tbody>
</table>

Output

<table>
<thead>
<tr>
<th>n-gram</th>
<th>p</th>
</tr>
</thead>
<tbody>
<tr>
<td>of</td>
<td>0.110</td>
</tr>
<tr>
<td>one of</td>
<td>0.233</td>
</tr>
<tr>
<td>are one of</td>
<td>0.447</td>
</tr>
</tbody>
</table>
Summary

Compute interpolated modified Kneser-Ney without pruning in
Four streaming passes and three sorts.

How do we make this efficient?
Streaming Framework

Memory is divided into blocks. Blocks are recycled.

Lazily Merge Input

Adjust Counts

Sort Block

Write to Disk

Prepare for next step.
Adjusted Counts Detail

Lazily merge counts in suffix order

Adjust counts

Sort each block in context order

Write to disk

Each vertex is a thread $\implies$ Simultaneous disk and CPU.
Experiment: Toolkit Comparison

- **Task**: Build an unpruned 5-gram language model
- **Data**: Subset of English ClueWeb09 (webpages)
- **Machine**: 64 GB RAM
- **Output Format**: Binary (or ARPA when faster)

IRST disk: 3-way split. Peak RAM of any one process (as if run serially).
Berkeley: Binary search for minimum JVM memory.
This Work 3.9G

SRI compact

SRI disk

SRI

RAM (GB)

Tokens (millions)

Estimating

Streaming and Sorting

Experiments
This Work 3.9G

IRST

SRI compact

SRI disk

SRI

IRST disk

Tokens (millions)
This Work 3.9G
This Work 3.9G

IRST

IRST disk

MIT

SRI compact

SRI disk

B

Diagram showing the relationship between Wall time (hours) and Tokens (millions) for different systems. The graph compares the performance of IRST, IRST disk, SRI disk, SRI compact, SRI, MIT, and This Work 3.9G in terms of wall time required for a given number of tokens.
CPU time (hours) vs Tokens (millions)

- IRST disk
- SRI disk
- SRI compact
- MIT
- This Work 3.9G

Experiments

Estimating

Streaming and Sorting
Scaling

<table>
<thead>
<tr>
<th>This Work</th>
<th>Tokens</th>
<th>Smoothing</th>
<th>Machines</th>
<th>Days</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>126 billion</td>
<td>Kneser-Ney</td>
<td>1</td>
<td>2.8</td>
</tr>
</tbody>
</table>

Counts

<table>
<thead>
<tr>
<th>This Work 126B</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pruned Google 1T</td>
<td>14m</td>
<td>315m</td>
<td>977m</td>
<td>1,313m</td>
<td>1,176m</td>
</tr>
</tbody>
</table>

(This work used a machine with 140 GB RAM and a RAID5 array.)
## Scaling

| This Work | 126 billion | Kneser-Ney | 1 | 2.8 | 2013 |
| Google    | 31 billion  | Kneser-Ney | 400 | 2 | 2007 |
| Google    | 1800 billion| Stupid     | 1500 | 1 | 2007 |

### Counts

<table>
<thead>
<tr>
<th>This Work 126B</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pruned Google 1T</td>
<td>14m</td>
<td>315m</td>
<td>977m</td>
<td>1,313m</td>
<td>1,176m</td>
</tr>
</tbody>
</table>

(This work used a machine with 140 GB RAM and a RAID5 array.)
WMT 2013 Results

1. Compress the big LM to 676 GB
2. Decode with 1 TB RAM
3. Make three WMT submissions

<table>
<thead>
<tr>
<th></th>
<th>Czech–English</th>
<th>French–English</th>
<th>Spanish–English</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Rank</td>
<td>BLEU</td>
<td>Rank</td>
</tr>
<tr>
<td>This Work</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>28.16</td>
<td>1</td>
</tr>
<tr>
<td>Google</td>
<td>2–3</td>
<td>27.11</td>
<td>2–3</td>
</tr>
<tr>
<td>Baseline</td>
<td>3–5</td>
<td>27.38</td>
<td>2–3</td>
</tr>
</tbody>
</table>

Rankings?

Pairwise significant above baseline
Build language models with user-specified RAM

kheafield.com/code/kenlm/

```
bin/lmplz -o 5 -S 10G <text >arpa
```

**Future Work**

- Interpolating models trained on separate data
- Pruning
- CommonCrawl corpus
Calculating Discounts

Summary statistics are collected while adjusting counts:
\[ s_n(a) = \text{number of } n\text{-grams with adjusted count } a. \]
Calculating Discounts

Summary statistics are collected while adjusting counts:

\[ s_n(a) = \text{number of } n\text{-grams with adjusted count } a. \]

\[
\text{discount}_n(a) = a - \frac{(a + 1)s_n(1)s_n(a + 1)}{(s_n(1) + 2s_n(2))s_n(a)}
\]

Chen and Goodman
Calculating Discounts

Summary statistics are collected while adjusting counts:
\[ s_n(a) = \text{number of } n\text{-grams with adjusted count } a. \]

Chen and Goodman discount
\[ \text{discount}_n(a) = a - \frac{(a + 1)s_n(1)s_n(a + 1)}{(s_n(1) + 2s_n(2))s_n(a)} \]

Use discount\(_n(3)\) for counts above 3.