A tutorial on the IRSTLM library

Nicola Bertoldi
FBK-irst, Trento, Italy

Berlin, May 17th 2008
Outline

• introduction to LM
• introduction to IRSTLM library
• space optimization
• distributed LM training
• support for chunk-based translation

Credits: M. Cettolo and M. Federico (FBK-irst, Trento)
N-gram LMs

The purpose of LMs is to compute the probability $\Pr(w_1^T)$ of any sequence of words $w_1^T = w_1 \ldots, w_t, \ldots, w_T$. The probability $\Pr(w_1^T)$ can be expressed as:

$$\Pr(w_1^T) = \Pr(w_1) \prod_{t=2}^{T} \Pr(w_t | h_t)$$

where $h_t = w_1, \ldots, w_{t-1}$ indicates the history of word $w_t$.

- $\Pr(w_t | h_t)$ become difficult to estimate as the sequence of words $h_t$ grows.
- we approximate by defining equivalence classes on histories $h_t$.
- $n$-gram approximation let each word depend on the most recent $n - 1$ words:

$$h_t \approx w_{t-n+1} \ldots w_{t-1}.$$
Data sparseness

Even estimating \( n \)-gram probabilities is not a trivial task:

- **high number of parameters**: e.g. a 3-gram LM with a vocabulary of 1,000 words requires, in principle, to estimate \( 10^9 \) probabilities!
- **data sparseness** of real texts: i.e. most of correct \( n \)-grams are rare events
- **smoothing** or **discounting**: frequency are not reliable

**Discount** relative frequency to assign some positive prob to every possible \( n \)-gram

\[
0 \leq f^*(w \mid x y) \leq f(w \mid x y) \quad \forall x y w \in V^3
\]

**Redistribution** of the zero-frequency probability \( \lambda(x y) \) over the set of words never observed after history \( x y \) proportional to \( p(w \mid y) \)

\[
\lambda(x y) = 1.0 - \sum_{w \in V} f^*(w \mid x y),
\]
Smoothing Schemes

*Discounted frequency* $f^*(w \mid x y)$ and redistribution of the *zero-frequency probability* $\lambda(x y)$ can be combined by:

- **Interpolation**, i.e. sum up the two approximations:

  $$p(w \mid x y) = f^*(w \mid x y) + \lambda(x y)p(w \mid y).$$

- **Back-off**, i.e. select the most significant approximation available:

  $$p(w \mid x y) = \begin{cases} f^*(w \mid x y) & \text{if } f^*(w \mid x y) > 0 \\ \alpha_{xy} \lambda(x y)p(w \mid y) & \text{otherwise} \end{cases}$$

  where $\alpha_{xy}$ is an appropriate *normalization term*
Smoothing Methods

- **Witten-Bell estimate** [Witten & Bell, 1991]
  \[ \lambda(xy) \propto n(xy) \] i.e. \# different words observed after \( xy \) in the training data:
  \[ \lambda(xy) =_{def} \frac{n(xy)}{c(xy) + n(xy)} \]
  which gives:
  \[ f^*(w \mid xy) = \frac{c(xyw)}{c(xy) + n(xy)} \]

- **Absolute discounting** [Ney & Essen, 1991]
  subtract constant \( \beta \) (\( 0 < \beta < 1 \)) from all observed \( n \)-gram counts
  \[ f^*(w \mid xy) =_{def} \max \left\{ \frac{c(xyw) - \beta}{c(xy)}, 0 \right\} \]
  which gives:
  \[ \lambda(xy) = \beta \frac{n(xy)}{c(xy)} \]

- **Kneser-Ney smoothing** [Kneser & Ney, 1995]
  Absolute discounting with *corrected counts* \( c'(yw) \) for lower order \( n \)-grams

- **Improved Kneser-Ney** [Chen & Goodman, 1998]
  Use *specific discounting coefficients* \( \beta = \beta(c(xyw)) \) for rare \( n \)-grams
Large Scale Language Models

- Availability of large scale corpora has pushed research toward using huge LMs
- At 2006 NIST WS best systems used LMs trained on at least 1.6G words
- Google presented results using a 5-gram LM trained on 1.3T words
- Handling of such huge LMs with available tools (e.g. SRILM) is prohibitive if you use standard computer equipment (4 up to 8Gb of RAM)
- Trend of technology is towards distributed processing using PC farms

We developed IRSTLM, a LM library addressing these needs
IRSTLM library

- **open-source** LGPL library under sourceforge.net
- full integration into the Moses SMT Toolkit and FBK-irst’s speech decoder
- different smoothing criteria in an interpolation scheme
- training of huge LMs
- support for chunk-based translation

- **space optimization**
- **distributed training on single machine or SGE queue**
- caching of LM calls
Space optimization

- $n$-gram collection uses dynamic storage to encode counters
- probs and back-off weights are quantized
- LM data structure is loaded on demand

[Federico & Cettolo, ACL-SMT '07]
Data Structure to Collect N-grams

• Dynamic prefix-tree data structure
• Successor lists are allocated on demand through memory pools
• Storage of counts from 1 to 6 bytes, according to max value
• Permits to manage few huge counts, such as in the google n-grams
Data Structure to Compute LM Probs

• First used in *CMU-Cambridge LM Toolkit* (Clarkson and Rosenfeld, 1997)
• Slower access but less memory than structure used by *SRILM Toolkit*
• *IRSTLM* can compress probs and back-off weights into 1 byte (instead of 4)!
Compression Through Quantization

How does quantization work?
1. Partition observed probabilities into regions (clusters)
2. Assign a code and probability value to each region (codebook)
3. Encode the probabilities of all observations (quantization)

We investigate two quantization methods:
- Lloyd's K-Means Algorithm
  - first applied to LM for ASR by [Whittaker & Raj, 2000]
  - computes clusters minimizing average distance between data and centroids
- Binning Algorithm
  - first applied to term-frequencies for IR by [Franz & McCarley, 2002]
  - computes clusters that partition data into uniformly populated intervals

Notice: a codebook of $n$ centers means a quantization level of $\log_2 n$ bits.
LM Quantization

- **Codebooks**
  - One codebook for each word and back-off probability level
  - For instance, a 5-gram LM needs in total 9 codebooks
  - Use codebook of at least 256 entries for 1-gram distributions

- **Motivation**
  - Distributions of these probabilities can be quite different
  - 1-gram distributions contain relatively few probabilities
  - Memory cost of a few codebooks is irrelevant.

- **Composition of codebooks**
  - LM probs are computed by multiplying entries of different codebooks

[Federico & Bertoldi, ACL-SMT '06]
**LM Quantization**

- Spanish-English translation on EPPS
- Lloyd and **binning** algorithms perform similarly
- **No loss in performance with 8 bit quantization**
Moses calls to a 3-gram LM while decoding from German to English the text:

ich bin kein christdemokrat und glaube daher nicht an wunder . doch ich möchte dem europäischen parlament, so wie es gegenwärtig beschaffen ist, für seinen grossen beitrag zu diesen arbeiten danken.
LM Accesses by SMT Search Algorithm

- 1.7M calls only involving 120K different 3-grams
- Decoder tends to access LM n-grams in non-uniform, *highly localized patterns*
- First call of an n-gram is easily followed by other calls of the same n-gram
our LM structure permits to exploit so-called *memory mapped* file access

memory mapping permits to include a file in the address space of a process, whose access is managed as virtual memory

only memory pages (grey blocks) that are accessed by decoding are loaded
Performance

- Chinese-English task of NIST MT Evaluation Workshop 2006
- Large parallel corpus (85 Mw), 6.1M 5-grams
- English giga monolingual corpus (1.8 Gw), 289M 5-grams
- Moses decoder

<table>
<thead>
<tr>
<th>LM</th>
<th>format</th>
<th>quant</th>
<th>file size</th>
</tr>
</thead>
<tbody>
<tr>
<td>lrg</td>
<td>textual</td>
<td>n</td>
<td>855Mb</td>
</tr>
<tr>
<td></td>
<td></td>
<td>y</td>
<td>685Mb</td>
</tr>
<tr>
<td></td>
<td>binary</td>
<td>n</td>
<td>296Mb</td>
</tr>
<tr>
<td></td>
<td></td>
<td>y</td>
<td>178Mb</td>
</tr>
<tr>
<td>giga</td>
<td>textual</td>
<td>n</td>
<td>28.0Gb</td>
</tr>
<tr>
<td></td>
<td></td>
<td>y</td>
<td>21.0Gb</td>
</tr>
<tr>
<td></td>
<td>binary</td>
<td>n</td>
<td>8.5Gb</td>
</tr>
<tr>
<td></td>
<td></td>
<td>y</td>
<td>5.1Gb</td>
</tr>
</tbody>
</table>

- Binarization: 65-75% reduction
- Quantization: 20% reduction for textual, 40% for binary
- Overall: -80%

N. Bertoldi
IRSTLM Library
Berlin, May 17th 2008
### Performance

<table>
<thead>
<tr>
<th>LM</th>
<th>BLEU score</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>05 06 06 06</td>
<td>nw</td>
<td>ng</td>
<td>bn</td>
</tr>
<tr>
<td>lrg SRILM</td>
<td>27.3 29.4 23.7 27.2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lrg</td>
<td>27.3 29.1 23.6 27.1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q-lrg</td>
<td>27.3 29.0 23.2 27.0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lrg+giga</td>
<td>29.2 29.7 24.8 28.6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q-lrg+q-giga</td>
<td>29.0 29.8 24.8 28.2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LM</th>
<th>NIST score</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>05 06 06 06</td>
<td>nw</td>
<td>ng</td>
<td>bn</td>
</tr>
<tr>
<td>lrg SRILM</td>
<td>8.60 9.00 7.88 8.57</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lrg</td>
<td>8.60 9.03 7.85 8.55</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q-lrg</td>
<td>8.56 8.99 7.77 8.51</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lrg+giga</td>
<td>8.84 8.92 7.92 8.70</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q-lrg+q-giga</td>
<td>8.75 9.08 8.06 8.65</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- SRILM and IRSTLM compares well (different prob to OOV words)
- Quantization does not affect performance significantly
- Use of giga increases performance significantly
### Performance

<table>
<thead>
<tr>
<th>LM</th>
<th>process size</th>
<th>caching</th>
<th>dec. speed (src w/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>lrg SRILM</td>
<td>1.2Gb</td>
<td>-</td>
<td>13.33</td>
</tr>
<tr>
<td>lrg</td>
<td>619Mb</td>
<td>n</td>
<td>6.80</td>
</tr>
<tr>
<td></td>
<td>558Mb</td>
<td>y</td>
<td>7.42</td>
</tr>
<tr>
<td>q-lrg</td>
<td>507Mb</td>
<td>n</td>
<td>6.99</td>
</tr>
<tr>
<td></td>
<td>445Mb</td>
<td>y</td>
<td>7.52</td>
</tr>
<tr>
<td>lrg+giga</td>
<td>9.9Gb</td>
<td>n</td>
<td>3.52</td>
</tr>
<tr>
<td></td>
<td>2.1Gb</td>
<td>y</td>
<td>4.28</td>
</tr>
<tr>
<td>q-lrg+q-giga</td>
<td>6.8Gb</td>
<td>n</td>
<td>3.64</td>
</tr>
<tr>
<td></td>
<td>2.1Gb</td>
<td>y</td>
<td>4.35</td>
</tr>
</tbody>
</table>

- IRSTLM requires less memory than SRILM (558Mb vs. 1.1Gb) (10 vs. 20Gb???)
- IRSTLM is slower than SRILM (7.42 vs. 13.33)
- quantization slightly speeds up decoding
- caching speeds up decoding (8-9% on lrg, 20-21% on lrg+giga)
Distributed LM training

- **goal**: reduce time and fit n-gram statistics into memory

- **idea**: partition n-grams into $k$ parts, train $k$ LMs, recombine into one LM

- **problem**: probabilities of the n-gram $x y w$ depends on $x y$ (and $y w$)
  \[
p(w \mid x y) = f^*(w \mid x y) + \lambda(x y)p(w \mid y)\]

- **solution**:  
  - split n-grams into self-consistent subsets: containing all information needed to compute $f^*(w \mid x y)$ and $\lambda(x y)$  
  - use an intermediate data structure to store all $f^*$ and $\lambda$  
  - compute probabilities on the fly, $P(w \mid x y) = f^*(w \mid x y) + \lambda(x y)*P(w \mid y)$

- **self-consistency** depends on the smoothing method
Available smoothing for distributed LM training

- **Witten Bell**: each subset should contain all successors of an $n$-gram
  \[ f^*(w \mid xy) = \frac{c(xyw)}{c(xy) + n(xy)} \quad \text{and} \quad \lambda(xy) = \frac{n(xy)}{c(xy) + n(xy)} \]

- **Absolute discounting**: the same as Witten Bell
  \[ f^*(w \mid xy) = \max \left\{ \frac{c(xyw) - \beta}{c(xy)}, 0 \right\} \quad \text{and} \quad \lambda(xy) = \beta \frac{\sum_{w: c(xyw) > 1} 1}{c(xy)} \]

- **Improved Kneser-Ney**: possible (without corrected counts)
  \[ f^*(w \mid x y) = \frac{c(xyw) - \beta(c(xyw))}{c(xy)} \]
  \[ \beta(0) = 0, \quad \beta(1) = D_1, \quad \beta(2) = D_2, \quad \beta(c) = D_3+ \]
How to to distributed LM training: step 0

get a training corpus

<table>
<thead>
<tr>
<th>TRAIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>this should also be there is looking further .</td>
</tr>
<tr>
<td>this we shall be there is looking further .</td>
</tr>
<tr>
<td>so we shall be there is looking further .</td>
</tr>
<tr>
<td>this should also be there would be a little .</td>
</tr>
<tr>
<td>this should also be there looking further ahead .</td>
</tr>
<tr>
<td>it should also be there is looking further .</td>
</tr>
<tr>
<td>so we shall be there is looking further .</td>
</tr>
<tr>
<td>this should also be there would be little .</td>
</tr>
<tr>
<td>this we shall be there would be a little .</td>
</tr>
<tr>
<td>this should also be there is going further .</td>
</tr>
<tr>
<td>so we shall be there would be a little .</td>
</tr>
<tr>
<td>this we shall be there is looking further ahead .</td>
</tr>
<tr>
<td>so we shall be there is looking further ahead .</td>
</tr>
<tr>
<td>this we shall be there would be little .</td>
</tr>
<tr>
<td>this may be , there would be a little .</td>
</tr>
<tr>
<td>this should also be there is to further .</td>
</tr>
<tr>
<td>so we shall be there would be little .</td>
</tr>
<tr>
<td>this we shall be there is going further .</td>
</tr>
<tr>
<td>so we shall be there is going further .</td>
</tr>
<tr>
<td>it should also be there would be a little .</td>
</tr>
</tbody>
</table>
How to distributed LM training: step 1

extract the dictionary

```
dict -InputFile=TRAIN -OutputFile=DICT -Freq=yes -sort=no
```
How to to distributed LM training: step 2

split dictionary into
balanced n-gram prefix lists

split-dict.pl --input DICT --output DICT. --parts 3
How to distributed LM training: step 3

collect n-grams for each prefix list

```
ngt -InputFile=TRAIN -FilterDict=DICT.000 -NgramSize=3 -OutputFile=WWW.000 -OutputGoogleFormat=yes
```

```
DICT.000
DICTIONARY 0 5
this 12
should 8
also 8
be 28
there 20

DICT.001
DICTIONARY 0 5
is 12
looking 8
further 12
. 20
we 11

WWW.000
this should also 6
this we shall 5
this may be 1
should also be 8
also be there 8
be there is 12
be there would 7
be a little 5
be , there 3
there is looking 8
there is going 3
there is to 1
there would be 8

WWW.001
is looking further 8
is going further 3
is to further 1
looking further . 5
looking further ahead 3
further . this 3
further . so 5
further . it 1
further ahead . 3
. this should 5
. this we 5
. this may 1
. so we 6
. it should 2
we shall be 11
```

this should also be there is looking further .
this we shall be there is looking further .
so we shall be there is looking further .
this should also be there would be a little .
this should also be there is looking further ahead .
it should also be there is looking further .
so we shall be there is looking further .
this should also be there would be little .
this we shall be there would be a little .
this should also be there is going further .
so we shall be there would be a little .
this should also be there is going further .
so we shall be there would be little .
it should also be there would be a little .
```
estimate single LMs (f* and λ) for each prefix list

```
build-sublm.pl --size 3 --ngrams WWW.000 --sublm LM.000
  [--prune-singletons] [--kneser-ney|--witten-bell]
```
How to to distributed LM training: step 5

merge single LMs

merge-sublm.pl --size 3 --sublm LM -lm iARPA_LM.gz

N. Bertoldi

IRSTLM Library

Berlin, May 17th 2008
Further steps for LM training

• optional steps:
  – transform into ARPA format
    compile-lm iARPA_LM.gz ARPA_LM --text yes
    compile-lm iARPA_LM.gz /dev/stdout --text yes | gzip-c > ARPA_LM.gz
  – quantize
    quantize-lm LM QLM
  – binarize
    compile-lm iARPA_LM.gz ARPA_LM

• perform steps 1-5 at once with
  build-lm.sh -i TRAIN -n 3 -o iARPA_LM.gz -k 3 [-p]

• if SGE queue is available, run a parallel version
  build-lm-qsub.sh -i TRAIN -n 3 -o iARPA_LM.gz -k 3 [-p]
# Distributed Training on English Gigaword

<table>
<thead>
<tr>
<th>list index</th>
<th>dictionary size</th>
<th>number of 5-grams:</th>
<th>observed distinct non-singletons</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td>217M 44.9M 16.2M</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>11</td>
<td>164M 65.4M 20.7M</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>208M 85.1M 27.0M</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>44</td>
<td>191M 83.0M 26.0M</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>64</td>
<td>143M 56.6M 17.8M</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>137</td>
<td>142M 62.3M 19.1M</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>190</td>
<td>142M 64.0M 19.5M</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>548</td>
<td>142M 66.0M 20.1M</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>783</td>
<td>142M 63.3M 19.2M</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>1.3K</td>
<td>141M 67.4M 20.2M</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>2.5K</td>
<td>141M 69.7M 20.5M</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>6.1K</td>
<td>141M 71.8M 20.8M</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>25.4K</td>
<td>141M 74.5M 20.9M</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>4.51M</td>
<td>141M 77.4M 20.6M</td>
<td></td>
</tr>
<tr>
<td>total</td>
<td>4.55M</td>
<td>2.2G 951M 289M</td>
<td></td>
</tr>
</tbody>
</table>
Chunk-based translation

- improve syntactic coherence of output
- use **shallow syntax (chunks)** on the target side (NC, VC, ...)
  SRC: Mein Freund wäscht sein neues Auto.
  TRG: (My friend|NC) (is washing|VC) (his new car|NC) (.|PNC)
- enlarge context: 3 chunks cover the full output

- Moses can not manage asynchronous factors (yet)
- split chunks into micro-chunks, X(, X+, X), X
  TRG: My|NP( friend|NP) is|VP( washing|VP) his|NP( new|NP+ car|NP) .|PNC
- train TM model with micro-chunks, LM model with chunks
- Moses generates translation options with micro-chunks

- **how to get chunk-based LM prob from micro-chunks strings?**
Chunk-based LM

- shrink sequence of micro-chunks into sequence of chunks
- use simple rules:
  \[ X \leftarrow X \]
  \[ X( X) \leftarrow X \]
  \[ X( X + \ldots X) \leftarrow X \]
- \[ P(My \text{ friend is washing his new car .}) = P("My") \ldots P("." | "new car") \]
- \[ P(NP( NP) VP( VP) NP( NP+ NP) PNC) \]
- \[ P(NP VP NP PNC) = P(NP) P(VP | NP) P(NP | NP VP) P(PNC | VP NC) \]
Thank you!

and use IRSTLM!
References
