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:

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

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

- $\Pr(w_t \mid h_t)$ 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

$$0 \leq f^*(w \mid x\ y) \leq f(w \mid x\ y) \qquad \forall x\ y\ w \in V^3$$

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 α_{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:

$$\lambda(xy) =_{def} \frac{n(xy)}{c(xy) + n(xy)} \quad \text{which gives:} \quad f^*(w \mid xy) = \frac{c(xyw)}{c(xy) + n(xy)}$$

- **Absolute discounting** [Ney & Essen, 1991]
  subtract constant β $(0 < \beta \leq 1)$ from all observed n-gram counts

$$f^*(w \mid xy) =_{def} \max\left\{\frac{c(xyw) - \beta}{c(xy)}, 0\right\} \quad \text{which gives} \quad \lambda(xy) = \beta \, \frac{n(xy)}{c(xy)}$$

- **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

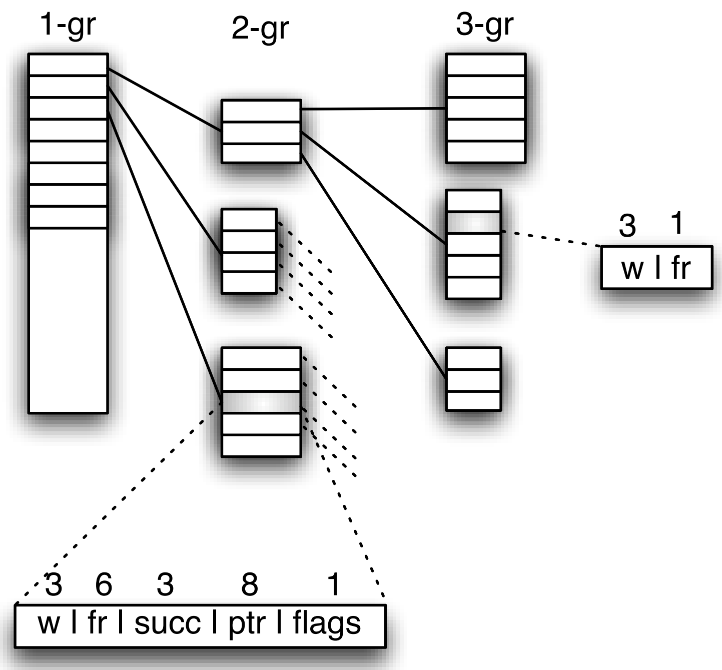
- **space optimization**
- **distributed training on single machine or SGE queue**
- caching of LM calls

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

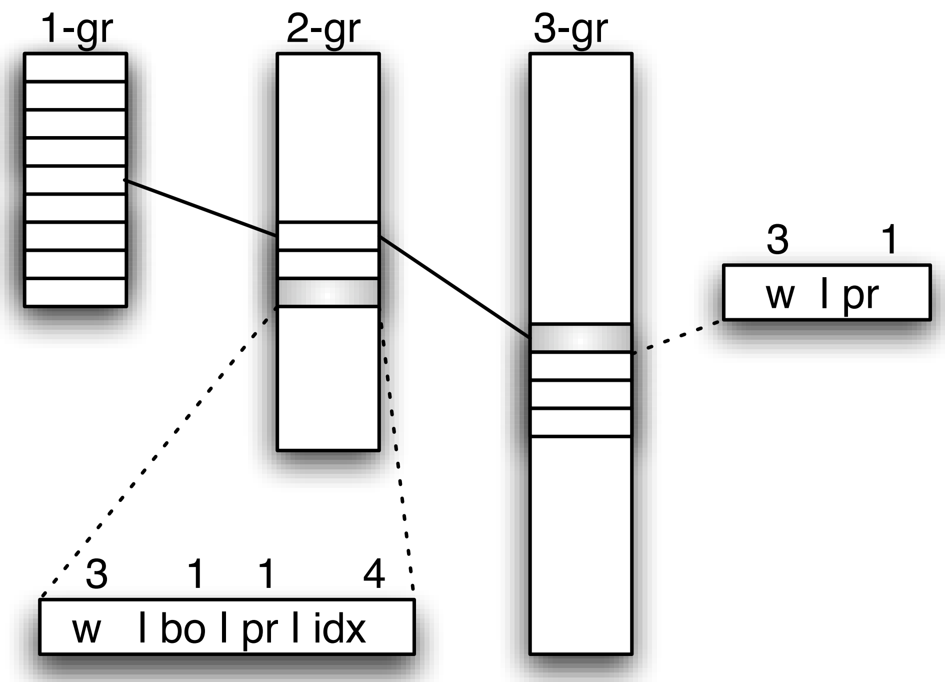
  [Federico & Cettolo, ACL-SMT '07]

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

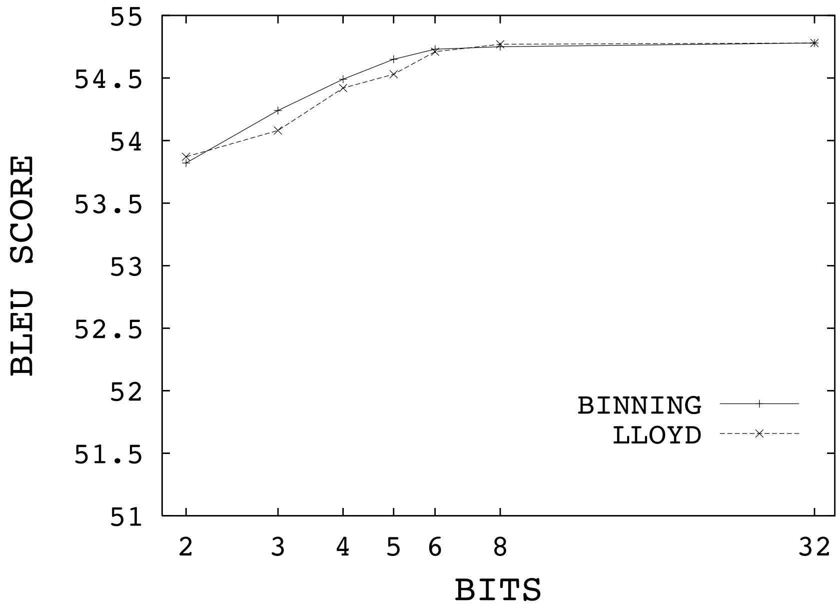
Notice: a codebook of n centers means a *quantization level* of $\log_2 n$ bits.

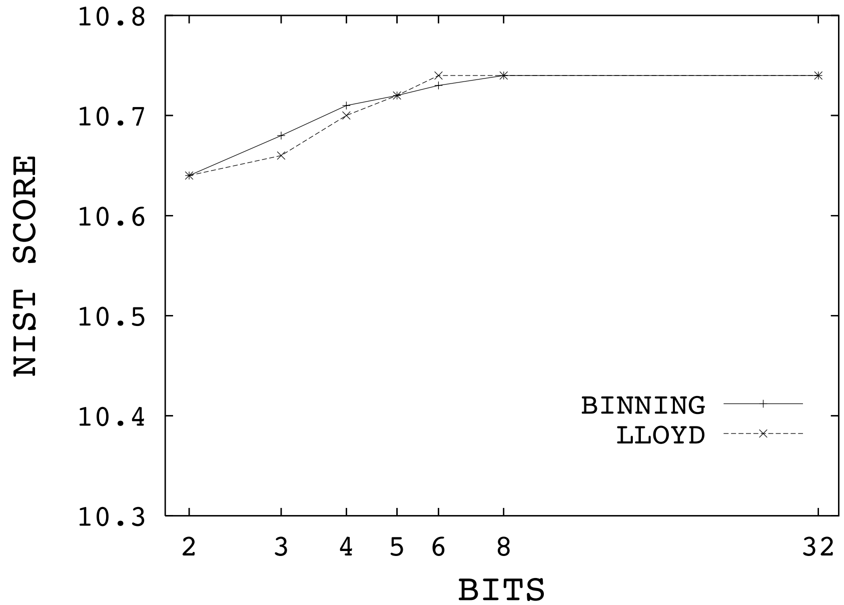
LM Quantization

- **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]

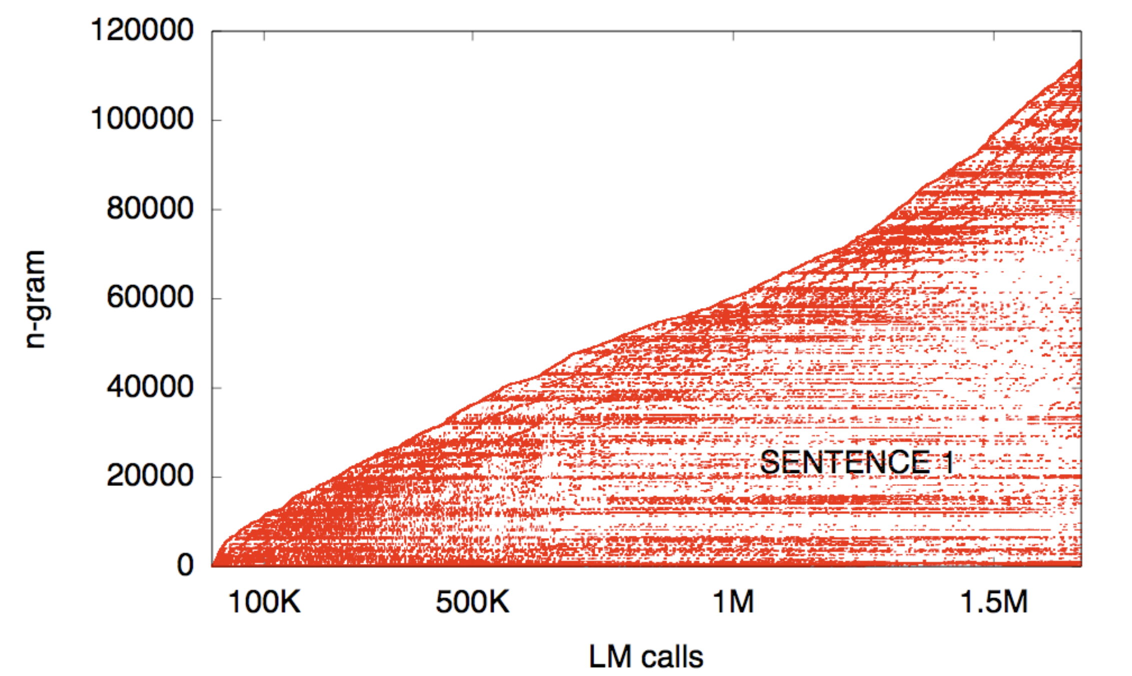
LM Quantization





- Spanish-English translation on EPPS
- Lloyd and **binning** algorithms perform similarly
- **No loss in performance with 8 bit quantization**

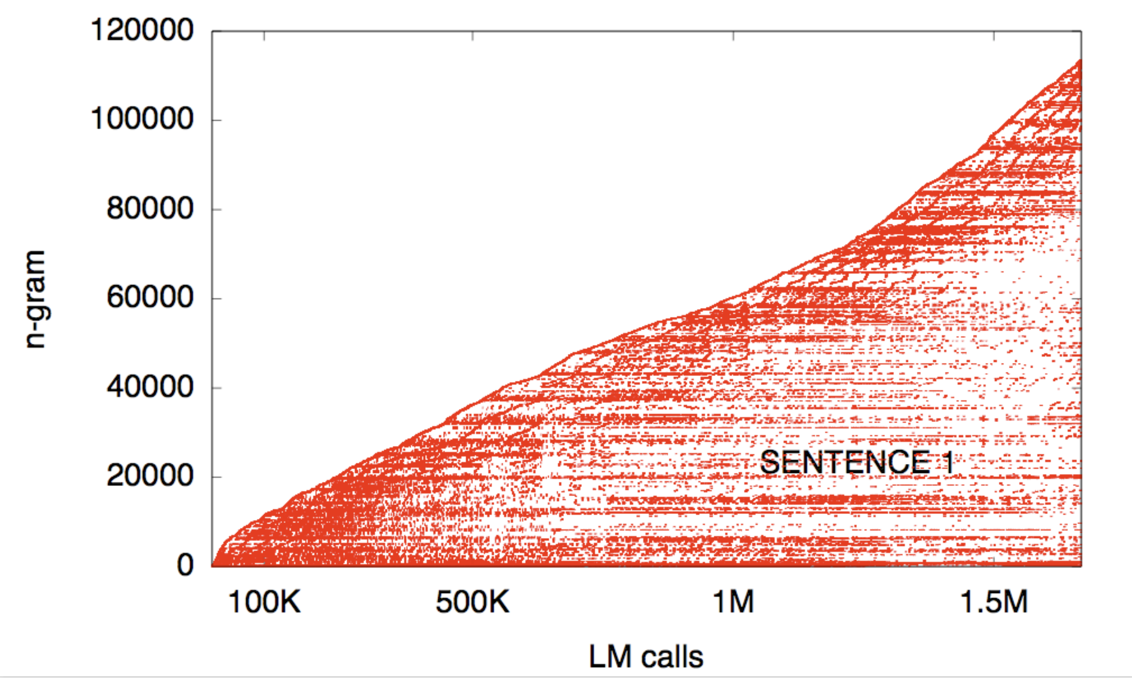
LM Accesses by SMT Search Algorithm



Moses calls to a 3-gram LM while decoding from German to English the text:

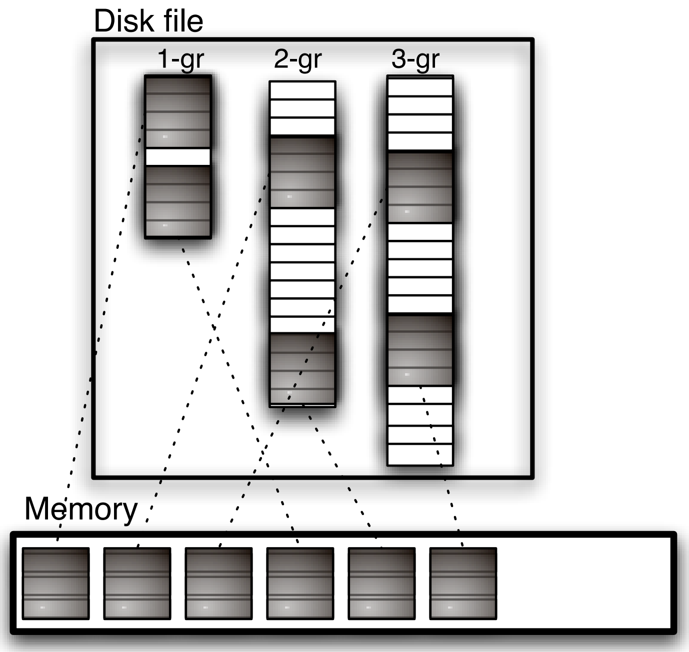
`ich bin kein christdemokrat und glaube daher nicht an wunder . doch ich möchte dem europäischen parlament , so wie es gegenwürtig beschaffen ist , für seinen grossen beitrag zu diesen arbeiten danken.`

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

Memory Mapping of LM on Disk



- 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

LM	format	quant	file size
lrg	textual	n	855Mb
		y	685Mb
	binary	n	296Mb
		y	178Mb

LM	format	quant	file size
giga	textual	n	28.0Gb
		y	21.0Gb
	binary	n	8.5Gb
		y	5.1Gb

- binarization: 65-75% reduction
- quantization: 20% reduction for textual, 40% for binary
- overall: -80%

Performance

LM	BLEU score			
	05	06 nw	06 ng	06 bn
lrg SRILM	27.3	29.4	23.7	27.2
lrg	27.3	29.1	23.6	27.1
q-lrg	27.3	29.0	23.2	27.0
lrg+giga	29.2	29.7	24.8	28.6
q-lrg+q-giga	29.0	29.8	24.8	28.2

LM	NIST score			
	05	06 nw	06 ng	06 bn
lrg SRILM	8.60	9.00	7.88	8.57
lrg	8.60	9.03	7.85	8.55
q-lrg	8.56	8.99	7.77	8.51
lrg+giga	8.84	8.92	7.92	8.70
q-lrg+q-giga	8.75	9.08	8.06	8.65

- SRILM and IRSTLM compares well (different prob to OOV words)
- quantization does not affect performance significantly
- use of `giga` increases performance significantly

Performance

LM	process size virtual	resident	caching	dec. speed (src w/s)
lrg SRILM	1.2Gb	1.1Gb	-	13.33
lrg	619Mb	558Mb	n	6.80
			y	7.42
q-lrg	507Mb	445Mb	n	6.99
			y	7.52
lrg+giga	9.9Gb	2.1Gb	n	3.52
			y	4.28
q-lrg+q-giga	6.8Gb	2.1Gb	n	3.64
			y	4.35

- 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 xyw depends on xy (and yw)
  $p(w \mid x\ y) = f^*(w \mid x\ y) + \lambda(x\ y) p(w \mid y)$
- **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 λ
  - 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
  $f^*(w \mid xy) = \frac{c(xyw)}{c(xy)+n(xy)}$ and $\lambda(xy) = \frac{n(xy)}{c(xy)+n(xy)}$

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

- **Improved Kneser-Ney**: possible (without corrected counts)
  $f^*(w \mid x\ y) = \frac{c(xyw)-\beta(c(xyw))}{c(xy)}$
  $\beta(0) = 0$, $\beta(1) = D_1$, $\beta(2) = D_2$, $\beta(c) = D_{3+}$

How to to distributed LM training: step 0

get a training corpus

TRAIN
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 we shall be there is looking further ahead . so we shall be there is looking further ahead . this we shall be there would be little . this may be , there would be a little . this should also be there is to further . so we shall be there would be little . this we shall be there is going further . so we shall be there is going further . it should also be there would be a little .

How to to distributed LM training: step 1

extract the dictionary

TRAIN

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 we shall be there is looking further ahead .
so we shall be there is looking further ahead .
this we shall be there would be little .
this may be , there would be a little .
this should also be there is to further .
so we shall be there would be little .
this we shall be there is going further .
so we shall be there is going further .
it should also be there would be a little .

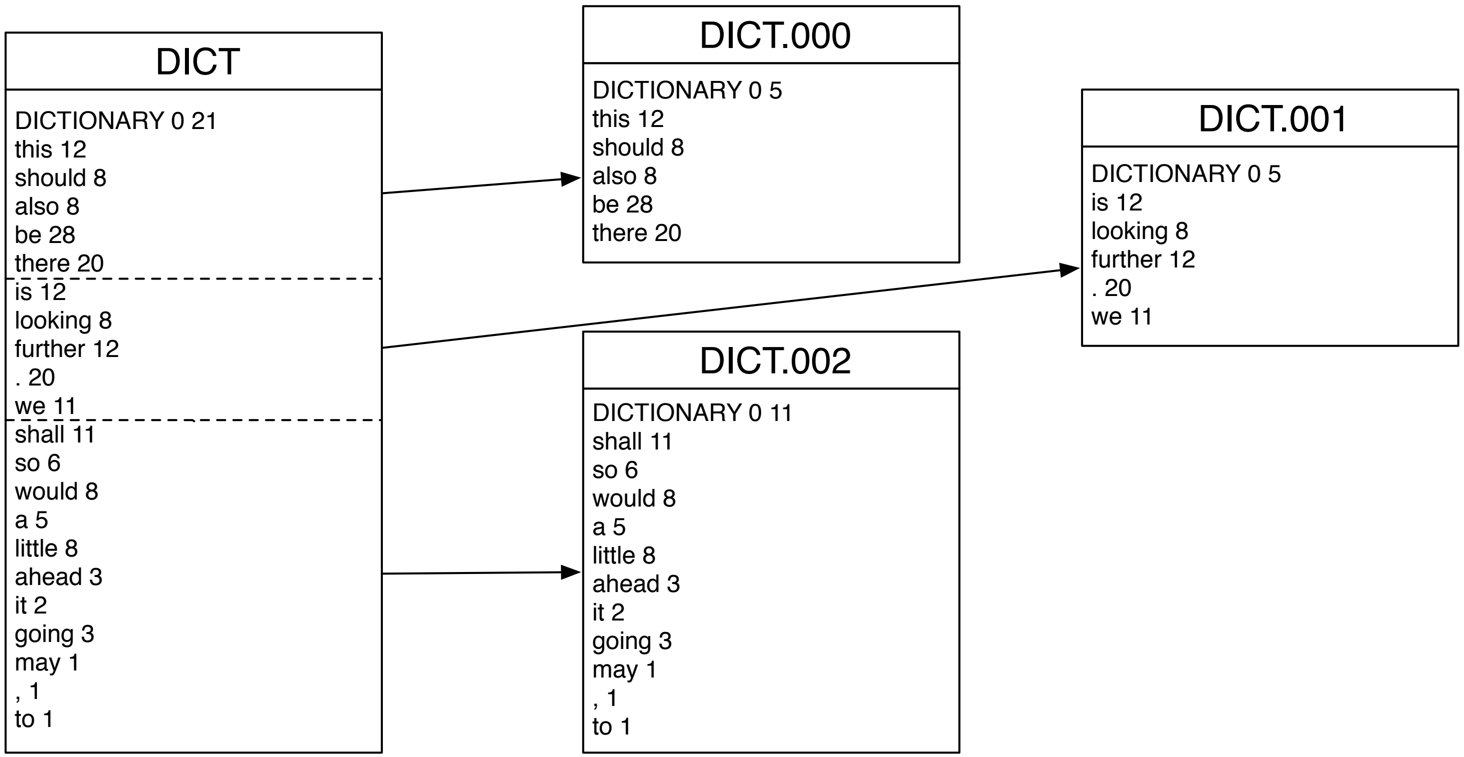
DICT

DICTIONARY 0 21
this 12
should 8
also 8
be 28
there 20
is 12
looking 8
further 12
. 20
we 11
shall 11
so 6
would 8
a 5
little 8
ahead 3
it 2
going 3
may 1
, 1
to 1

```
dict -InputFile=TRAIN -OutputFile=DICT -Freq=yes -sort=no
```

How to to distributed LM training: step 2

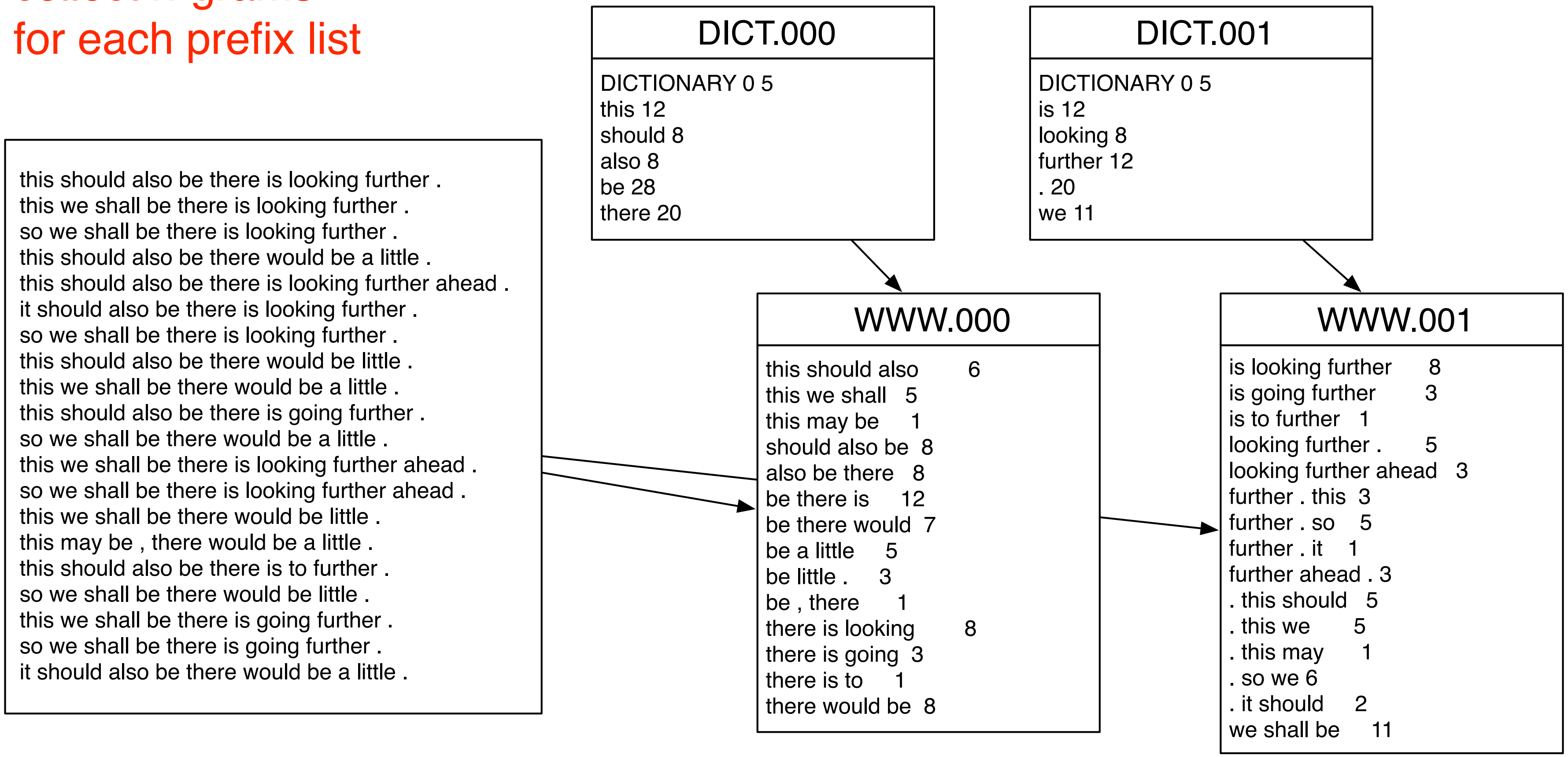
split dictionary into
balanced n-gram prefix lists




```
split-dict.pl --input DICT --output DICT. --parts 3
```

How to 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
```

How to to distributed LM training: step 4

estimate single LMs (f* and λ)
for each prefix list

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 little . 3
be , there 1

LM.000.1gr

12 this -0.698970
8 should -0.954243
8 also -0.954243
28 be -0.903090
20 there -1.041393

LM.000.2gr

-0.397940 this should -0.845098
-0.477121 this we -0.778151
-1.176091 this may 0.000000
-0.051153 should also -0.954243
-0.051153 also be -0.954243
-0.226396 be there -1.021189
-0.806180 be a -0.778151
-1.028029 be little -0.602060
-1.505150 be , 0.000000
-0.263241 there is -0.574031
-0.439333 there would -0.954243

LM.000.3gr

-0.066947 this should also
-0.079181 this we shall
-0.051153 should also be
-0.051153 also be there
-0.243038 be there is
-0.477121 be there would
-0.079181 be a little
-0.124939 be little .
-0.273001 there is looking
-0.698970 there is going
-0.051153 there would be

```
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

LM.000.1gr

```
12 this -0.698970
8 should -0.954243
8 also -0.954243
28 be -0.903090
20 there -1.041393
```

LM.001.1gr

```
.......
```

LM.001.1gr

```
.......
```

LM.000.2gr

```
-0.397940 this should -0.845098
-0.477121 this we -0.778151
-1.176091 this may 0.000000
-0.051153 should also -0.954243
-0.051153 also be -0.954243
-0.226396 be there -1.021189
-0.806180 be a -0.778151
-1.028029 be little -0.602060
-1.505150 be , 0.000000
-0.263241 there is -0.574031
-0.439333 there would -0.954243
```

LM.001.2gr

```
.......
```

LM.002.2gr

```
.......
```

LM.000.3gr

```
-0.066947 this should also
-0.079181 this we shall
-0.051153 should also be
-0.051153 also be there
-0.243038 be there is
-0.477121 be there would
-0.079181 be a little
-0.124939 be little .
-0.273001 there is looking
-0.698970 there is going
-0.051153 there would be
```

LM.001.3gr

```
.......
```

LM.002.3gr

```
.......
```

iArpa_LM

```
iARPA
\data\
ngram 1= 22
ngram 2= 32
ngram 3= 31
\1-grams:
-1.249669 this -0.698970
-1.409369 should -0.954243
-1.409369 also -0.954243
-0.901214 be -0.903090
-1.041393 there -1.041393
.........
\2-grams:
-0.397940 this should -0.845098
-0.477121 this we -0.778151
-1.176091 this may 0.000000
-0.051153 should also -0.954243
-0.051153 also be -0.954243
.........
\3-grams:
-0.066947 this should also
-0.079181 this we shall
-0.051153 should also be
-0.051153 also be there
-0.243038 be there is
.........
\end\
```

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

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

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

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+ ... X) $\leftarrow$ X
- P(My friend is washing his new car .) $= P$("My") ... 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

Federico, Bertoldi. "How Many Bits Are Needed To Store Probabilities for Phrase-Based Translation?". ACL Workshop on SMT. New York City, NY, US, 2006.

Federico, Marcello, Mauro Cettolo, "Efficient Handling of N-gram Language Models for Statistical Machine Translation". ACL 2007 Workshop on SMT. Prague, Czech Republic, 2007