

IRSTLM Toolkit

Nicola Bertoldi
FBK-irst Trento, Italy

Fifth MT Marathon, Le Mans, France

13-18 September 2010

Outline

- introduction to LM toolkit
- ARPA file format for LM representation
- IRSTLM library
- memory and time optimization
- distributed LM training

Credits:

- M. Cettolo and M. Federico (FBK-irst, Trento)
- IRSTLM developers and users

Language Model toolkit

A Language Model toolkit should provide functionalities for (at least):

- estimating n -gram probabilities from a text corpus
- computing probability of an n -gram
- computing perplexity of a test sample

- (several) different smoothing criteria
- pruning techniques
- adaptation methods

Most known LM toolkits are:

- CMU/Cambridge: mi.eng.cam.ac.uk/prc14/toolkit.html
- SRILM: www.speech.sri.com/projects/srilm
- IRSTLM: hlt.fbk.eu/en/irstlm

ARPA File Format (srilm, irstlm)

Represents both interpolated and back-off n-gram LMs

- format: $\log(\text{smoothed-prob}) :: \text{n-gram} :: \log(\text{back-off weight})$
- computation: look first for smoothed-prob, otherwise back-off

```
\data\  
ngram 1 =      86700  
ngram 2 =     1948935  
ngram 3 =     2070512  
  
1-grams:  
-2.88382      !           -2.38764  
-2.94351     world       -0.514311  
-6.09691     guys          -0.15553  
...  
2-grams:  
-3.91009     world !         -0.351469  
-3.91257     hello world    -0.24  
-3.87582     hello guys     -0.0312  
..  
3-grams:  
-0.00108858  hello world !  
-0.000271867 , hi hello !  
...  
\end\  

```

Example 1:

$\log\text{Pr}(!|\text{hello world}) = -0.00108858$

Example 2:

$\log\text{Pr}(!|\text{hello guys}) = -0.0312 + \log\text{Pr}(!|\text{guys})$

$\log\text{Pr}(!|\text{guys}) = -0.15553 + \log\text{Pr}(!)$

$\log\text{Pr}(!) = -2.88382$

Large/Huge 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, 1up to4 cores)
~ 2G running words, ~ 200M 5-grams, ~ 9Gb RAM
- Trend of technology is towards:
 - distributed processing using PC farms
 - space and time optimization

IRSTLM addresses these needs, in a fully open source (Moses-like) framework

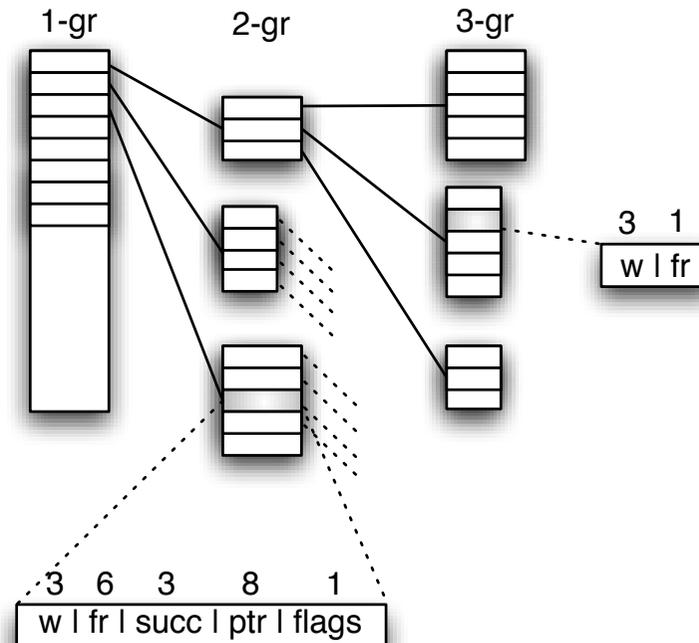
IRSTLM library

- open-source LGPL library (hlt.fbk.eu/en/irstlm)
 - full integration into Moses SMT Toolkit and FBK speech decoder
 - different smoothing criteria in an interpolation scheme: WB, AD, MKN
 - singleton pruning, adaptation, and internal/external interpolation
 - training of huge LMs
 - support for chunk-based translation
-
- memory optimization
 - speed optimization
 - distributed training on single machine or SGE queue

Memory optimization

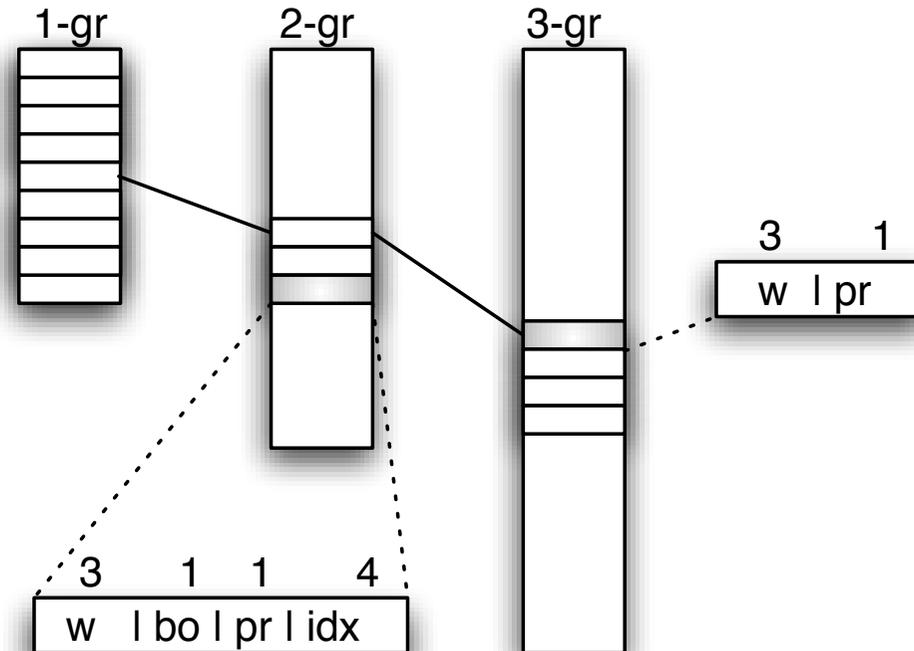
- dynamic storage to collect n -gram counts
- static storage to store n -gram probs
- quantization
- singleton pruning
- on-demand loading of LM

Data Structure to Collect N-grams



- **Dynamic** prefix-tree data structure
- Successor lists are allocated on demand through memory pools
 - specific successor lists for singletons
- 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 Store LM Probs



- **Static** data structure
- 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)!

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:

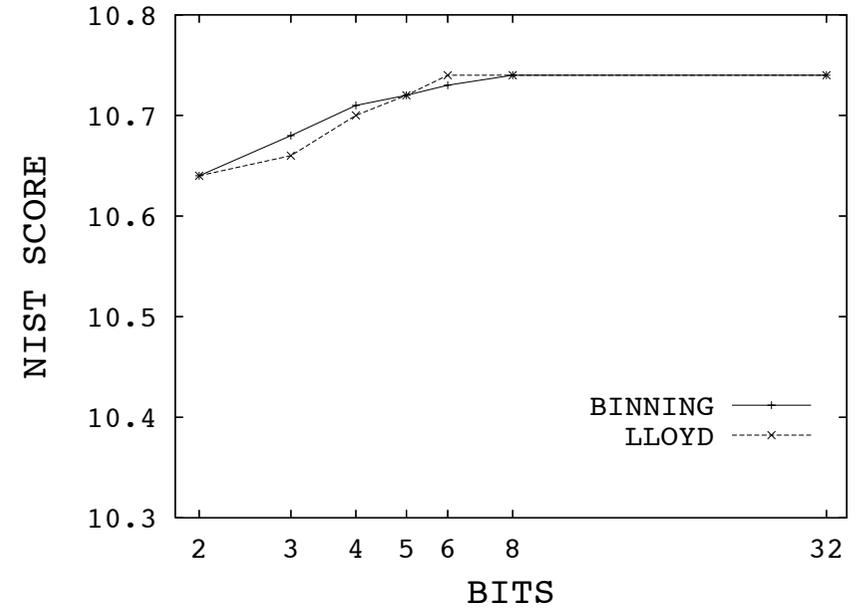
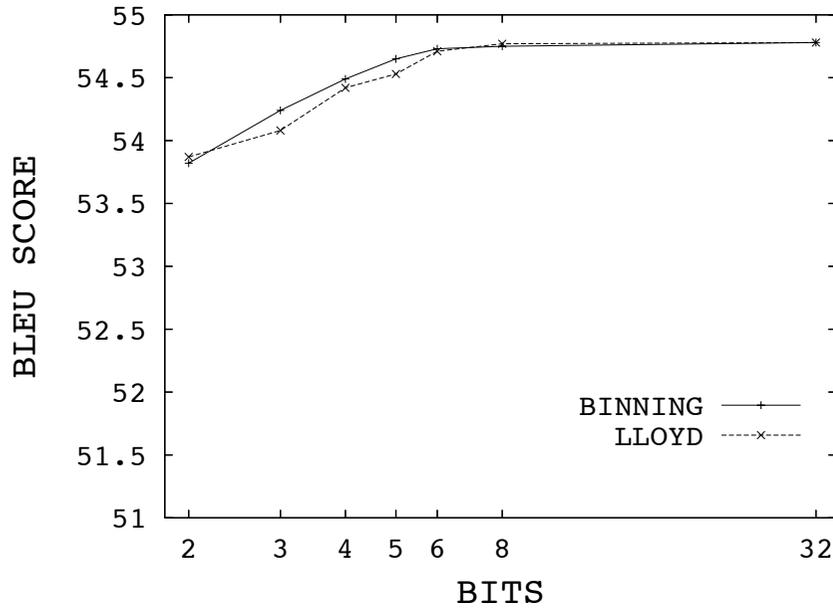
- **Lloyd's K-Means Algorithm**
 - first applied to LM for ASR by [Whittaker & Raj, 2000]
 - computes clusters minimizing average distance between data and centroids
- **Binning Algorithm**
 - first applied to term-frequencies for IR by [Franz & McCarley, 2002]
 - computes clusters that partition data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

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

Quantization

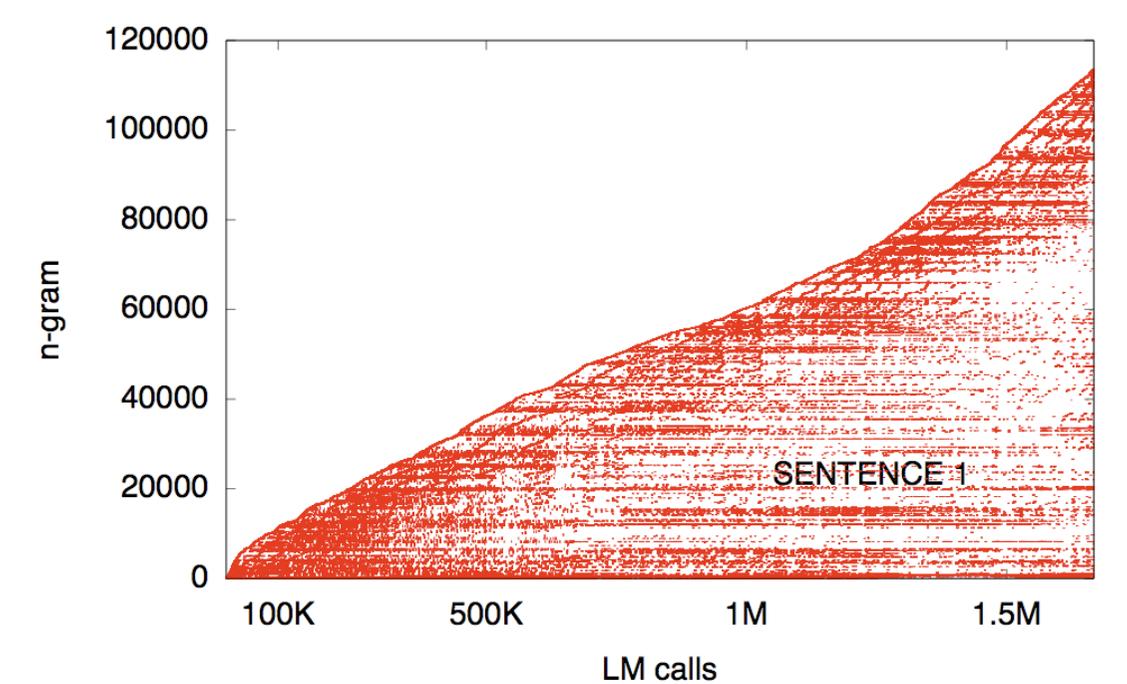
- **Codebooks**
 - One codebook for each word and back-off probability level
 - For instance, a 5-gram LM needs in total 9 codebooks
 - Use codebook of at least 256 entries for 1-gram distributions
- **Motivation**
 - Distributions of these probabilities can be quite different
 - 1-gram distributions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

Quantization



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

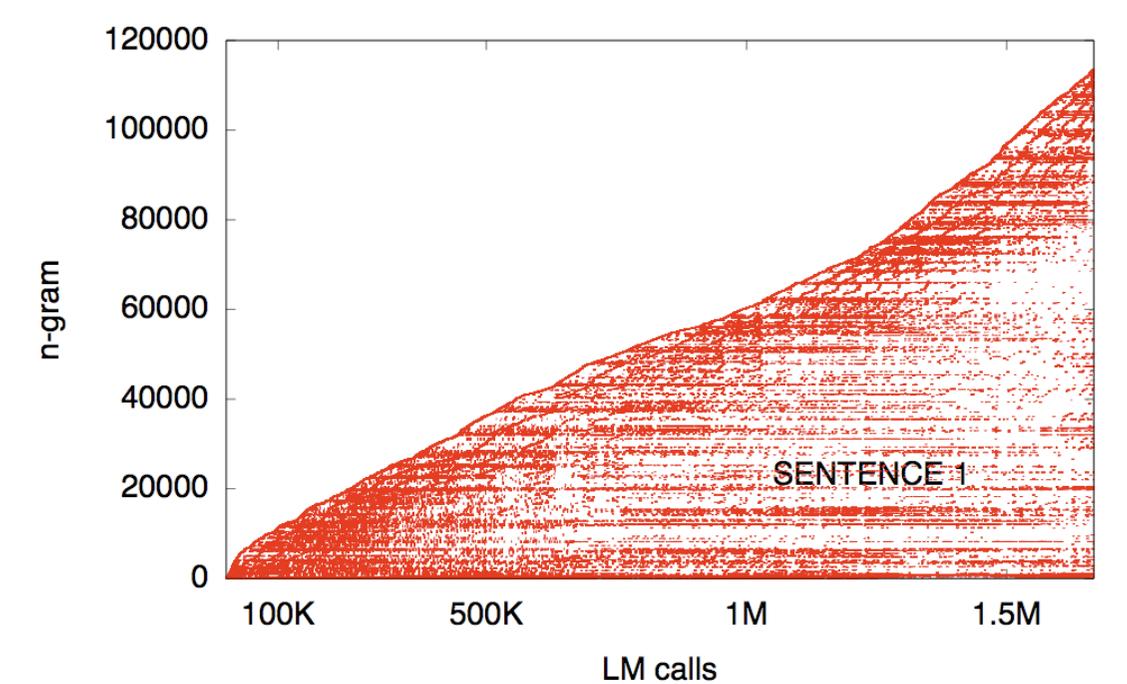
LM Accesses by SMT Search Algorithm



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

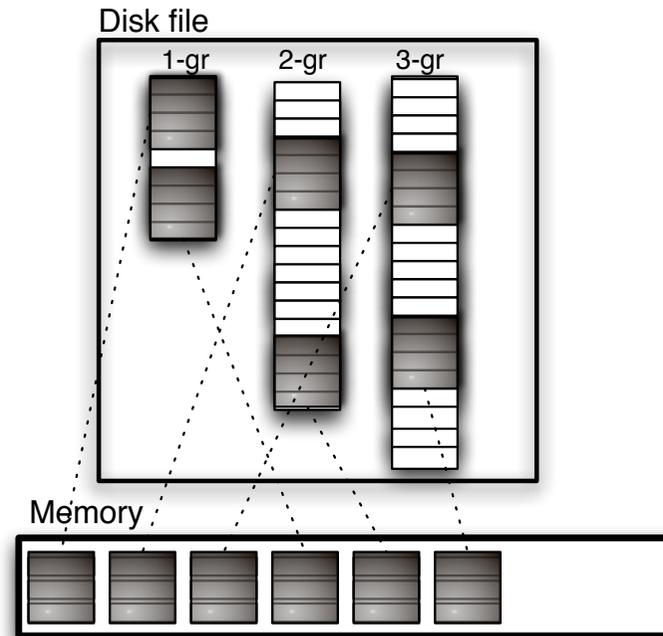
ich bin kein christdemokrat und glaube daher nicht an wunder . doch ich möchte dem europäischen parlament , so wie es gegenwärtig beschaffen ist , für seinen grossen beitrag zu diesen arbeiten danken.

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

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

Probability caching

- **Insight:**
 - during decoding, prob of the same n -gram is queried several time (14 on avg)
 - a LM call for an n -gram requires up to n accesses to the static data structure (in the worst case when no lower n -gram)
- **Solution:** caching
 - when an n -gram is queried for prob, check the cache before!
 - when not found, compute its probability and cache it (prob and state)

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	06	06
		nw	ng	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	06	06
		nw	ng	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		caching	dec. speed (src w/s)
	virtual	resident		
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)
- 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)

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

$$f^*(w | xy) = \frac{c(xyw)}{c(xy)+n(xy)} \text{ and } \lambda(xy) = \frac{n(xy)}{c(xy)+n(xy)}$$

- **Absolute discounting**: the same as Witten Bell

$$f^*(w | xy) = \max \left\{ \frac{c(xyw)-\beta}{c(xy)}, 0 \right\} \text{ and } \lambda(xy) = \beta \frac{\sum_{w:c(xyw)>1} 1}{c(xy)}$$

- **Improved Kneser-Ney**: possible (without corrected counts)

$$f^*(w | x y) = \frac{c(xyw)-\beta(c(xyw))}{c(xy)}$$

$$\beta(0) = 0, \beta(1) = D_1, \beta(2) = D_2, \beta(c) = D_{3+}$$

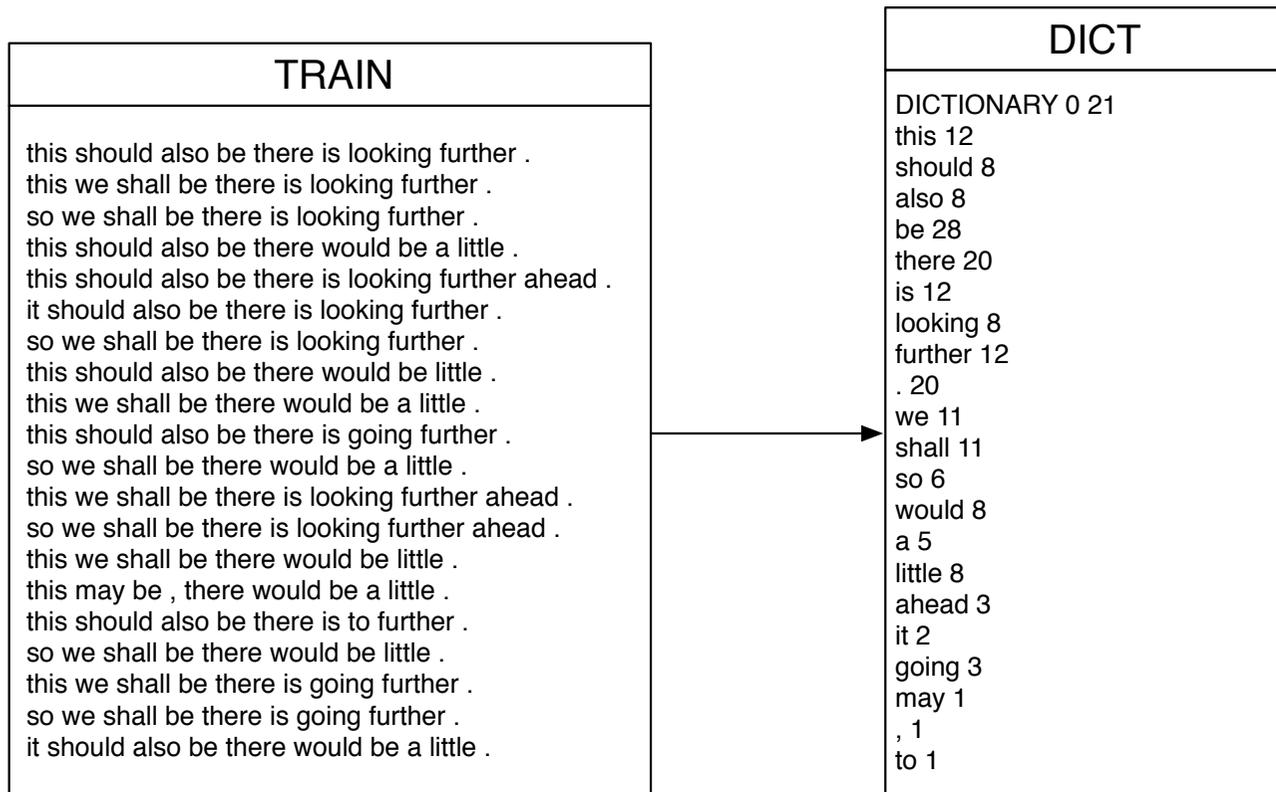
How to to distributed LM training: step 0

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

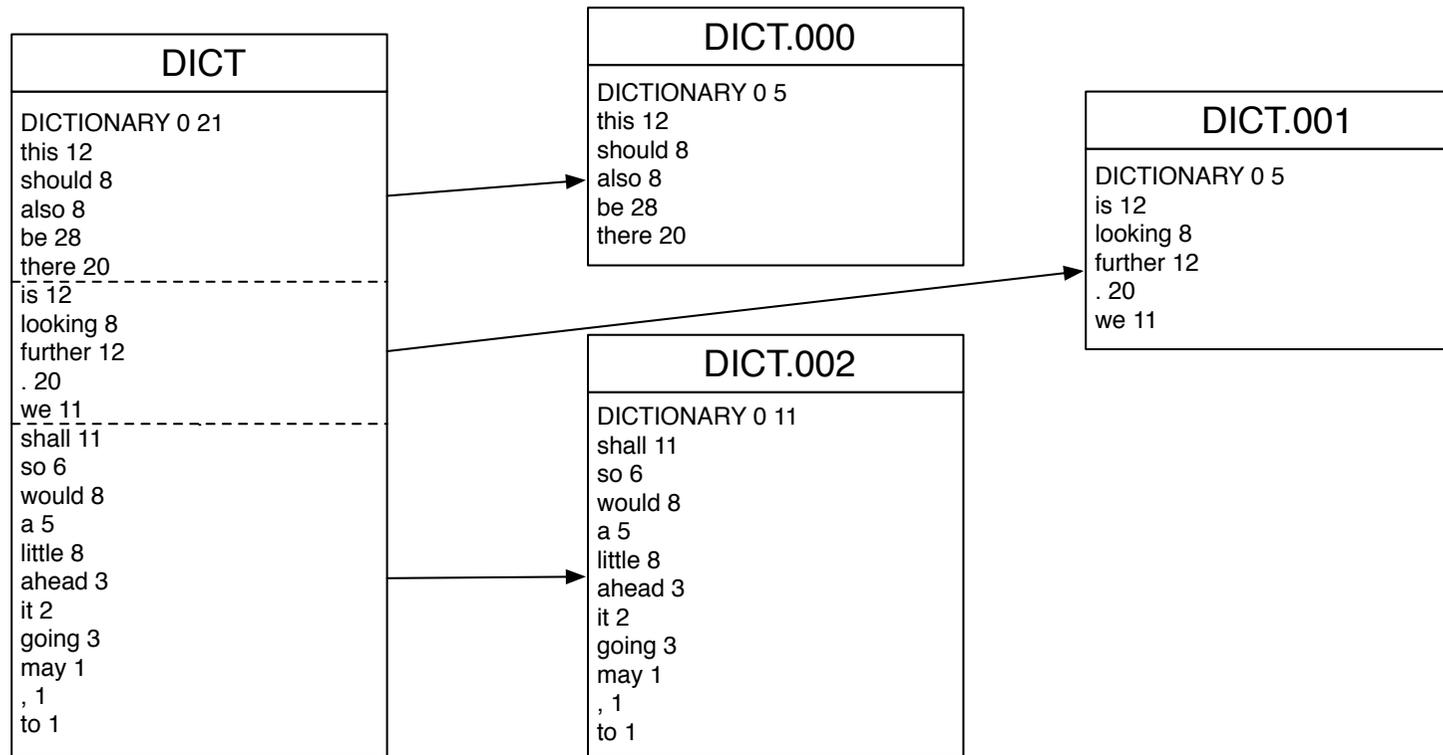
extract the dictionary



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

How to to distributed LM training: step 2

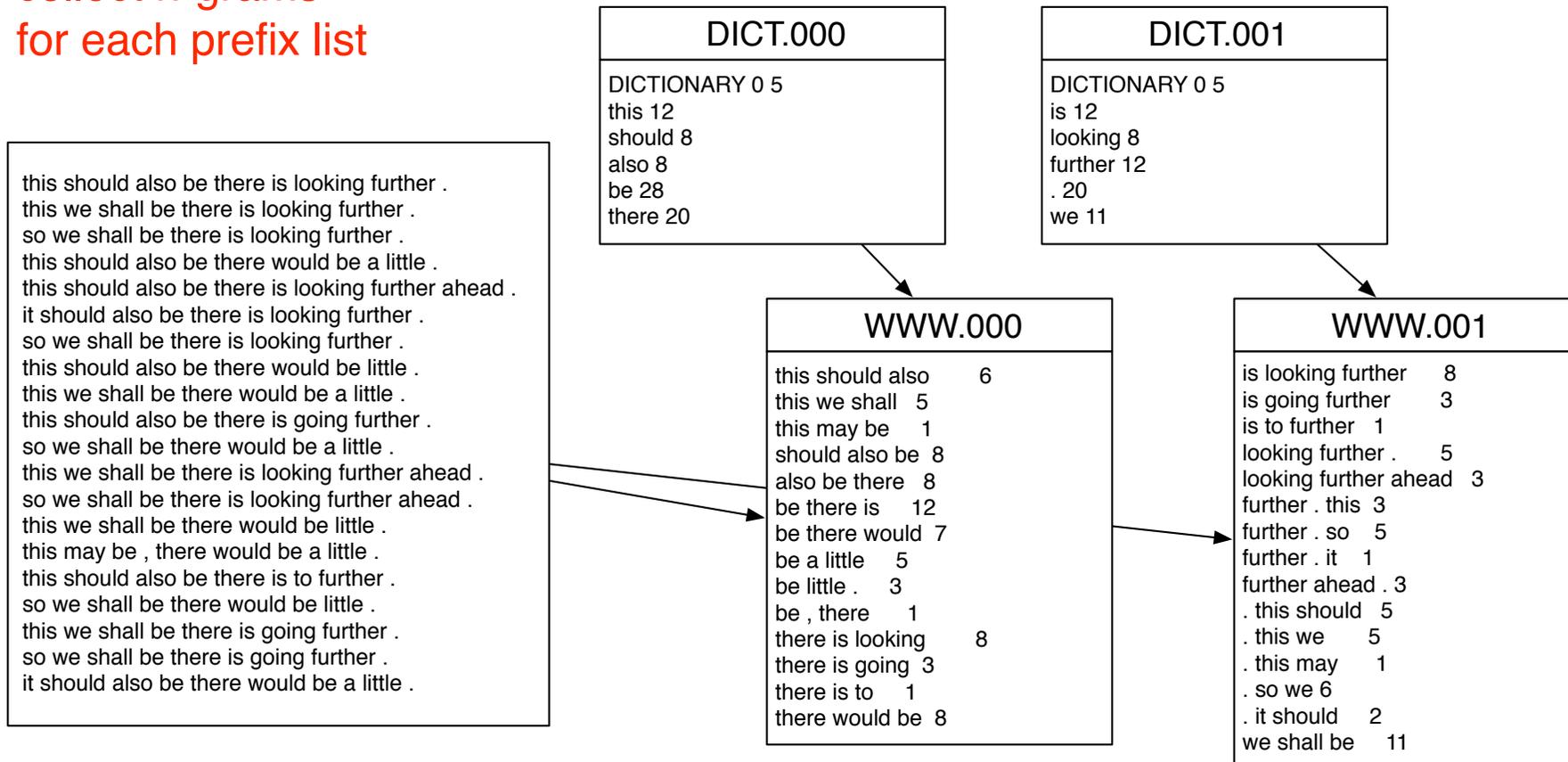
split dictionary into
balanced n-gram prefix lists



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

How to to distributed LM training: step 3

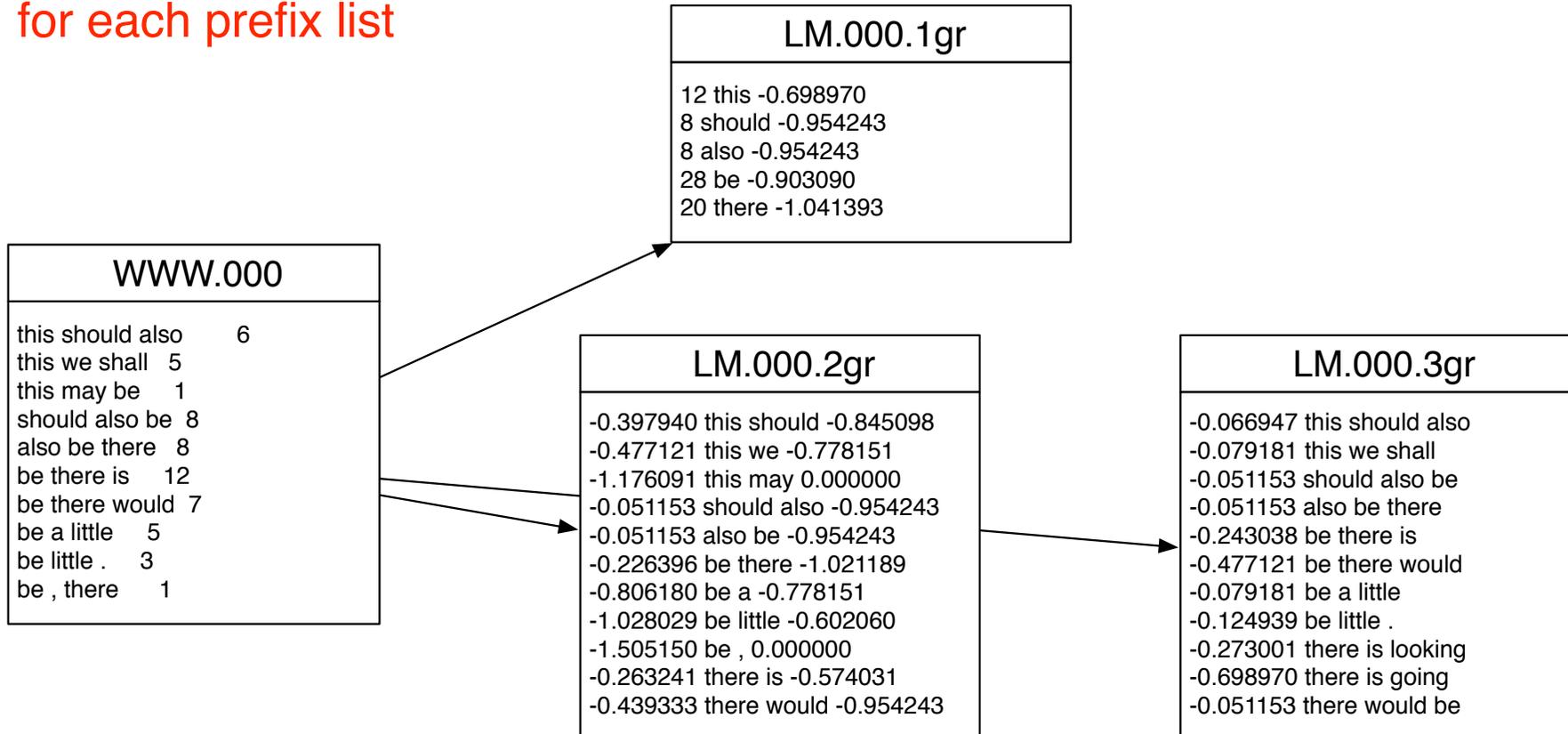
collect n-grams
for each prefix list



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

How to to distributed LM training: step 4

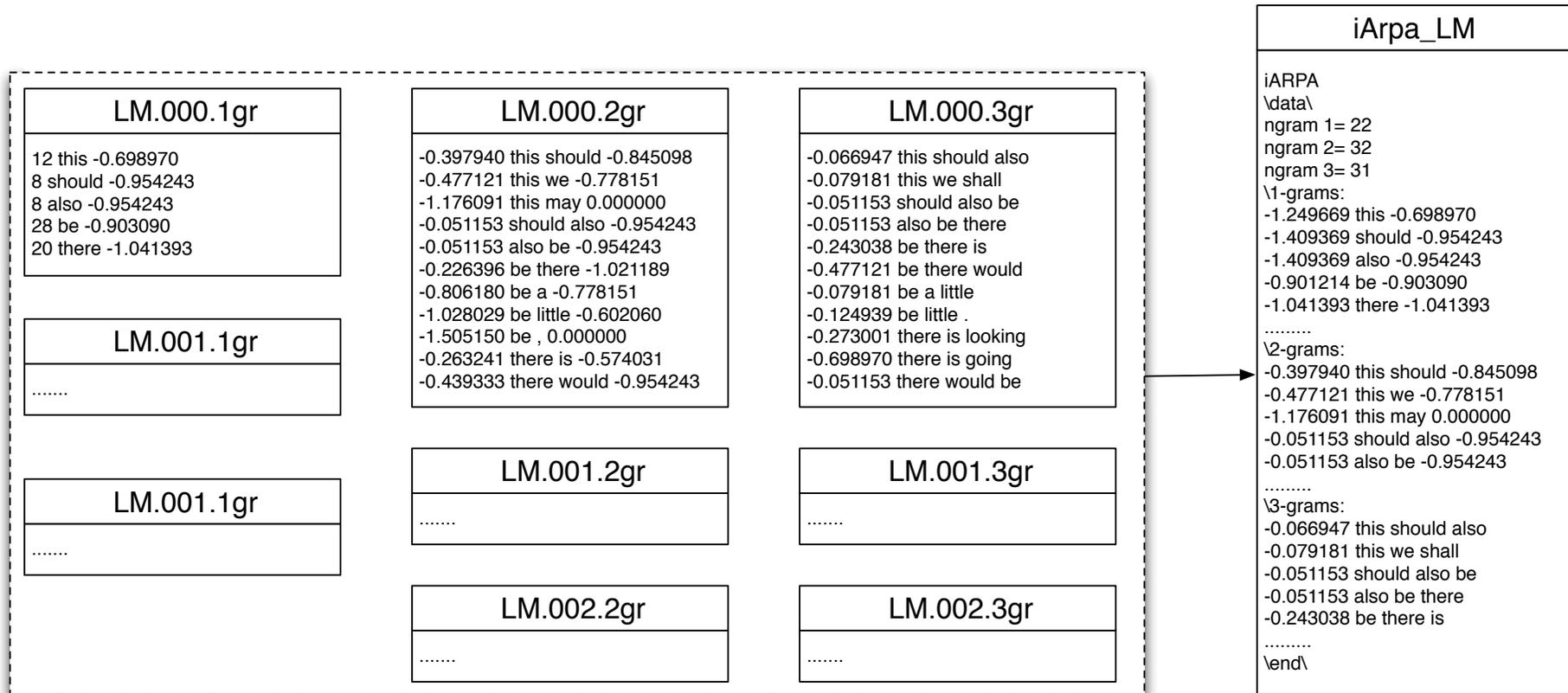
estimate single LMs (f^* and λ)
for each prefix list



```
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
```

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	number of 5-grams:		
	size	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:
 - $X \leftarrow X$
 - $X(X) \leftarrow X$
 - $X(X+ \dots X) \leftarrow X$
- $P(\text{My friend is washing his new car .}) = P(\text{"My"}) \dots P(\text{"."} \mid \text{"new car"})$
 $P(\text{NP(NP) VP(VP) NP(NP+ NP) PNC})$
 $P(\text{NP VP NP PNC}) = P(\text{NP}) P(\text{VP} \mid \text{NP}) P(\text{NP} \mid \text{NP VP}) P(\text{PNC} \mid \text{VP NC})$

Soon available

- faster caching
- thread-safe library
- faster access to the static LM data structure

Few hints about training

- pay attention when building the training data
 - homogeneity of training and test data
 - text normalization (dates, numbers, names, acronyms, etc.)
 - symbols for start and end sentence
- and when choosing the LM setting
 - data size
 - LM order
- to get the best tradeoff between
 - complexity
 - time/memory consumption
 - quality

References

- [Federico and Bertoldi, 2006] Federico, M. and Bertoldi, N. (2006). How many bits are needed to store probabilities for phrase-based translation? In Proc. ACL Workshop on SMT, pages 94–101, New York City.
- [Federico, and Cettolo, 2007] Federico, M. and Cettolo, M. (2007) Efficient Handling of N-gram Language Models for Statistical Machine Translation Proc. of ACL Workshop on SMT. pages 88–95, Prague, Czech Republic.

CMU/Cambridge: mi.eng.cam.ac.uk/prc14/toolkit.html

SRILM: www.speech.sri.com/projects/srilm

IRSTLM: hlt.fbk.eu/en/irstlm