Efficient Handling of n-gram Language Models for Statistical Machine Translation

M. Federico
FBK-irst Trento, Italy

Edinburgh, April 16, 2007

Summary

- Motivations
- Role of language model in SMT
- Introduction to n-gram LMs
  - Smoothing methods
  - LM representation/computation
- IRST LM Toolkit for Moses
  - Distributed estimation
  - Efficient data structures
  - Memory management
- Experiments
- Conclusions

Credits:

N. Bertoldi (FBK-irst), M. Cettolo (FBK-irst), C. Dayer (U. Maryland), H. Hoang (U. Edinburgh)

Motivation

N-gram LMs are major components of NLP systems, e.g. ASR and MT:

- 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 to to 8Gb of RAM)
- Trend of technology so far rewards distributing work on more PCs

We developed an alternative LM library addressing these needs

- IRSTLM is open-source Lesser GPL
- available and integrated into the Moses SMT Toolkit

Classical SMT Formulation

Let $\mathbf{f}$ be any text in the source language (French). The most probable translation is searched among texts $\mathbf{e}$ in the target language (English).

SMT used the following criterion:

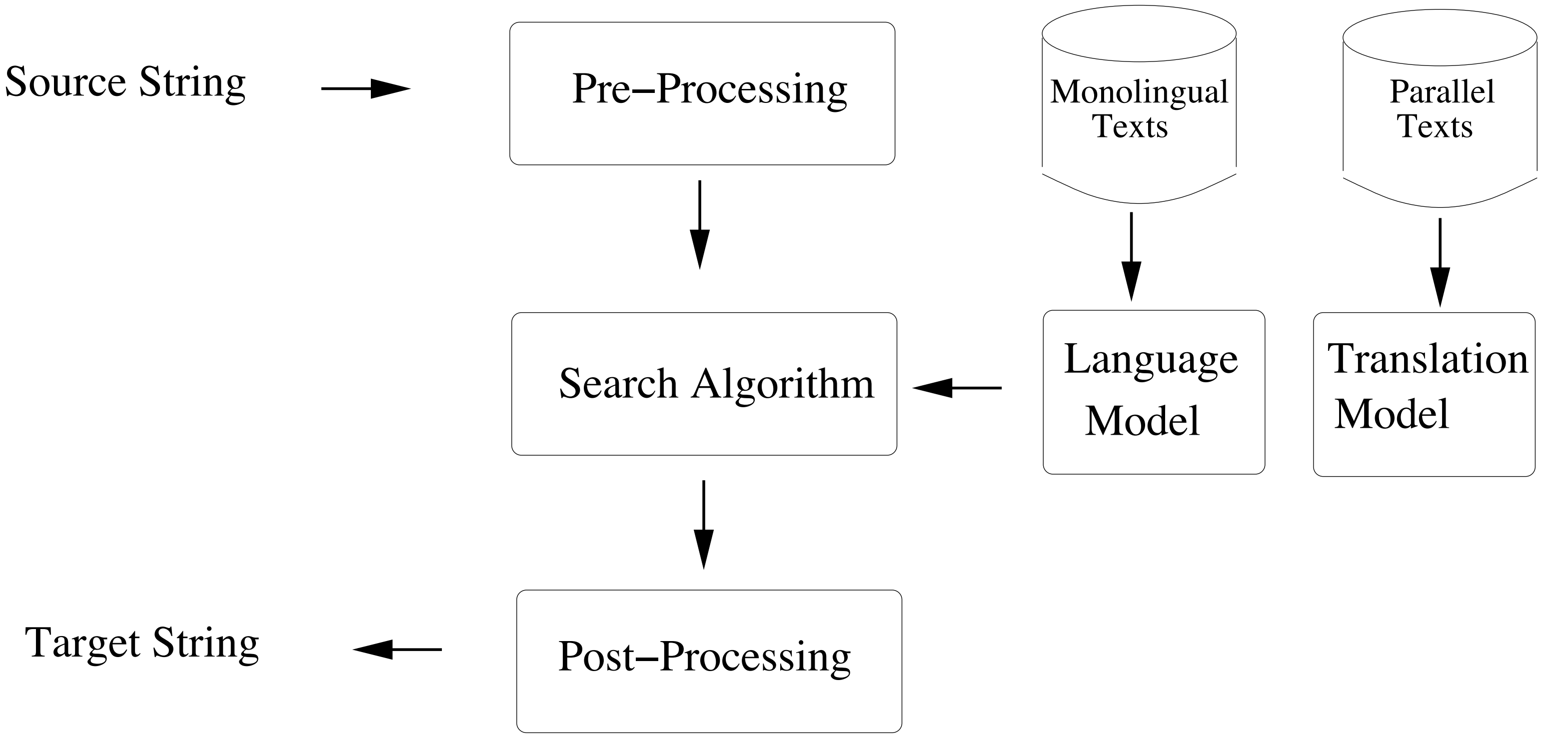
$$\mathbf{e}^* = \arg\max_{\mathbf{e}} \Pr(\mathbf{f} \mid \mathbf{e}) \Pr(\mathbf{e}) \tag{1}$$

The **computational problems of SMT**:

- *language modeling*: estimating the *language model probability* $\Pr(\mathbf{e})$
- *translation modeling*: estimating the *translation model probability* $\Pr(\mathbf{f} \mid \mathbf{e})$
- *search* problem: carrying out the optimization criterion (1)

Remark: in statistical MT all translation pairs are plausible, in principle.

Classical SMT Architecture



Log-linear phrase-based SMT

- Translation hypotheses are ranked by a **log-linear combination of statistics**:

$$rank_{\mathbf{e}} \ \max_{\mathbf{a}} \sum_i \lambda_i h_i(\mathbf{e}, \mathbf{f}, \mathbf{a})$$

$\mathbf{f}$ =*source*, $\mathbf{e}$ =*target*, $\mathbf{a}$ =*alignment*, and $h_i(\mathbf{e}, \mathbf{f}, \mathbf{a})$ =*feature functions*.

- Feature functions: **Language Model**, **Lexicon Model**, **Distortion Model**
- LM and TM consist of a huge number of *observations-value* pairs
- Example: *5-gram LM* – $h_i(\mathbf{e}, \mathbf{f}, \mathbf{a}) = \log \Pr(\mathbf{e})$
  – observations: 1-grams, 2-grams, 3-grams, 4-grams, 5-grams
  – values: log of cond. word probabilities, log of back-off weights
- Example: *Moses lexicon model*
  – observations: aligned phrase-pairs of length 1 to 8 words
  – values: log of dir/inv relative freq, dir/inv compositional logprobs

N-gram LM

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:

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

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:

$$h_t \approx w_{t-n+1} \ldots w_{t-1}. \tag{3}$$

Normalization Requirement

$$\sum_{T=1}^{\infty} \Pr(T) \sum_{w_1 \ldots w_T} \Pr(w_1, \ldots, w_T \mid T) = 1$$

N-gram LMs guarantee that probabilities sum up over one, for a given length T:

$$\begin{aligned}
\sum_{w_1 \ldots w_T} \prod_{t=1}^{T} \Pr(w_t \mid h_t) &= \sum_{w_1} \Pr(w_1) \sum_{w_2} \Pr(w_2 \mid h_1) \ldots \sum_{w_{T-1}} \Pr(w_{T-1} \mid h_{T-1}) \underbrace{\sum_{w_T} \Pr(w_T \mid h_T)}_{=1} \\
&= \sum_{w_1} \Pr(w_1) \sum_{w_2} \Pr(w_2 \mid h_1) \ldots \underbrace{\sum_{w_{T-1}} \Pr(w_{T-1} \mid h_{T-1})}_{=1} \cdot 1 \\
&= \ldots \\
&= \underbrace{\sum_{w_1} \Pr(w_1)}_{=1} \cdot 1 \ldots \cdot 1 \cdot 1 = 1 \qquad (4)
\end{aligned}$$

String Length Model

Hence we just need a length model $P(T)$

- **Exponential model** $p(T) = (a-1)a^{-T}$ with any $a > 1$, in fact:

$$\sum_{T=1}^{\infty} p(T) = (a-1)\sum_{T=1}^{\infty} a^{-T} = \frac{a-1}{a}\sum_{T=0}^{\infty}\left(\frac{1}{a}\right)^{T} = \frac{a-1}{a}\frac{1}{\left(1-\frac{1}{a}\right)} = \frac{a-1}{a-1} = 1 \quad (5)$$

  – Implemented in SMT by the *word penalty model*

- **Uniform model** over a range "of interest":

$$p(T) = \begin{cases} \frac{1}{T_{max}} & \text{if } 1 \leq T \leq T_{max} \\ 0 & \text{otherwise} \end{cases} \quad (6)$$

  – Used in SMT when no word penalty model is considered

N-gram LM and data sparseness

Even estimating n-gram probabilities may be 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*

Experimentally, in the 1.2Mw (million word) Lancaster-Oslo-Bergen corpus:

- more than 20% of bigrams and 60% of trigrams occur only once
- 85% of trigrams occur less than five times.
- expected chances of finding new 2-grams is 22%
- expected change of finding new 3-grams is 65%

We need frequency smoothing or discounting!

Frequency Discounting

Discount relative frequency to assign some positive prob to every possible n-gram

$$0 \leq f^*(w \mid h) \leq f(w \mid h) \qquad \forall hw \in V^n$$

The *zero-frequency probability* $\lambda(h)$, defined by:

$$\lambda(h) = 1.0 \; - \sum_{w \in V} f^*(w \mid h),$$

is *redistributed* over the set of words never observed after history h.

Redistribution is proportional to the less specific $n-1$-gram model $p(w \mid \bar{h})$.[1]

[1] Notice: $c(h) = 0$ implies that $\lambda(h) = 1$.

Smoothing Schemes

Discounting of $f(w \mid h)$ and redistribution of $\lambda(h)$ can be combined by:

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

$$p(w \mid h) = \begin{cases} f^*(w \mid h) & \text{if } f^*(w \mid h) > 0 \\ \alpha_h \lambda(h) p(w \mid \bar{h}) & \text{otherwise} \end{cases} \quad (7)$$

  where α_h is an appropriate *normalization term*[2]

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

$$p(w \mid h) = f^*(w \mid h) + \lambda(h) p(w \mid \bar{h}). \quad (8)$$

---

[2]
$$\alpha_h = \left(\sum_{w: f^*(w|h)=0} p(w \mid \bar{h}) \right)^{-1} = \left(1 - \sum_{w: f^*(w|h)>0} p(w \mid \bar{h}) \right)^{-1}$$

Smoothing Methods

- **Witten-Bell estimate** [Witten & Bell, 1991]
  $\lambda(h) \propto n(h)$ i.e. # different words observed after h in the training data:

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

- **Absolute discounting** [Ney & Essen, 1991]
  subtract constant β $(0 < \beta \leq 1)$ from all observed n-gram counts[3]

$$f^*(w \mid h) = \max\left\{\frac{c(hw)-\beta}{c(h)}, 0\right\} \text{ which gives } \quad \lambda(h) = \beta\frac{\sum_{w:c(h,w)>1} 1}{c(h)}$$

[3] $\beta \approx \frac{n_1}{n_1+2n_2} < 1$ where n_c is # of different n-grams which occurr c times in the training data.

Improved Absolute Discounting

- **Kneser-Ney smoothing** [Kneser & Ney, 1995]
  Absolute discounting with *corrected counts for lower order n-grams*. Rationale: the lower order frequency $f(\bar{h}, w)$ is made proportional to the number of different words that $(\bar{h}, w)$ follows.

  Example: let $c(\texttt{los}, \texttt{angeles}) = 1000$ and $c(\texttt{angeles}) = 1000 \longrightarrow$ corrected count is $c'(\texttt{angeles}) = 1$, i.e. unigram prob $p(\texttt{angeles})$ will be small.

- **Improved Kneser-Ney** [Chen & Goodman, 1998]
  In addition use *specific discounting coefficients* for rare n-grams:

$$f^*(w \mid h) = \frac{c(hw) - \beta(c(h, w))}{c(h)}$$

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

LM representation: ARPA File Format

Contains all the ingredients needed to compute LM probabilities:

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

logPr(!| hello edinburgh) = -0.0312 + logPr(!| edinburgh)

logPr(logPr(!| edinburgh) = -0.15553 - 2.88382

Moses Toolkit for Statistical MT

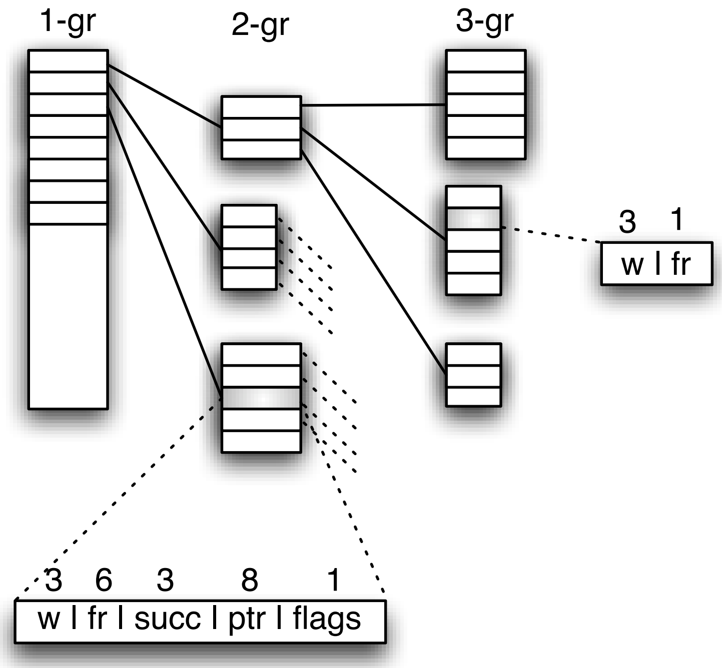
- Developed during **JHU Summer Workshop 2006**
  - U. Edinburgh, ITC-irst Trento, RWTH Aachen, U. Maryland, MIT, Charles University Prague
  - open source under Lesser GPL
  - available for Linux, Windows and Mac OS
  - www.statmt.org/moses
- **Main features**:
  - *translation of both text and CN inputs*
  - exploitation of more Language Models
  - lexicalized distortion model (only for text input, optional)
  - incremental pre-fetching of translation options from disk
  - *handling of huge LMs* (up to Giga words)
  - *on-demand and on-disk access to LMs* and LexMs
  - factored translation model (surface forms, lemma, POS, word classes, ...)

IRSTLM library (open source)

Important Features

- *Distributed training*
  - split dictionary into balanced n-gram prefix lists
  - collect n-grams for each prefix lists
  - estimate single LMs for each prefix list (approximation)
  - quickly merge single LMs into one ARPA file
- *Space optimization*
  - n-gram collection uses dynamic storage to encode counters
  - LM estimation just requires reading disk files
  - probs and back-off weights are quantized
  - LM data structure is loaded on demand
- *LM caching*
  - computations of probs, access to internal lists, LM states, ...

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

LM Estimation with Prefix Lists

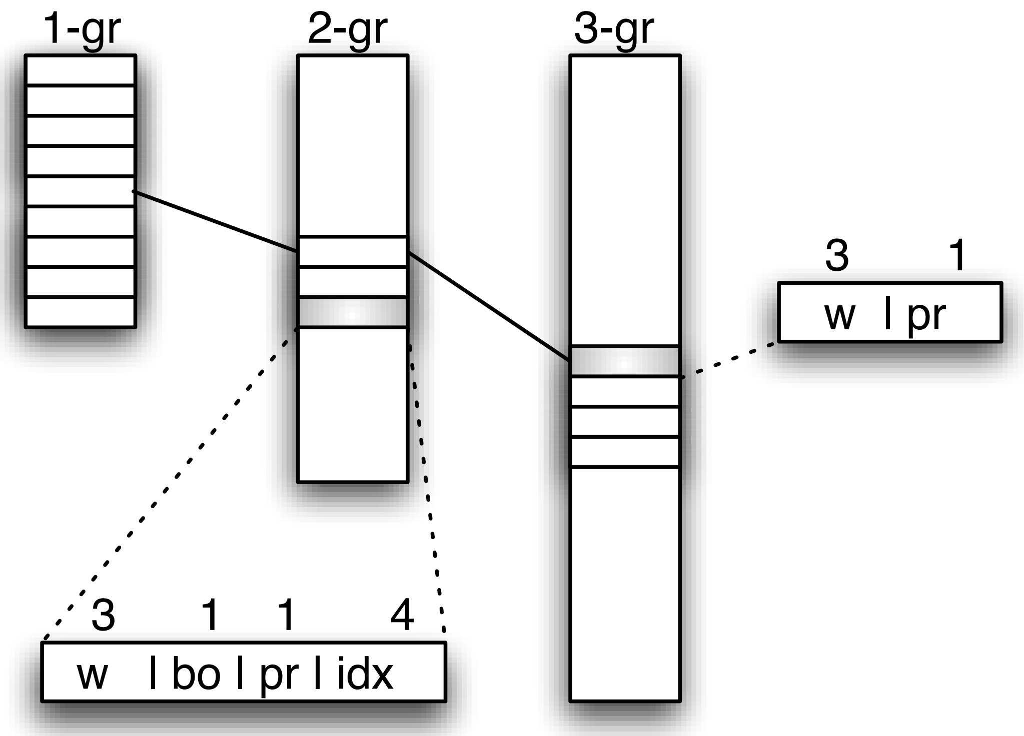
Smoothing of probs up from 2-grams is done separately on each subset of n-grams. Let (v, w, x, y, z) be a 5-gram :

- **Witten-Bell smoothing** (equivalent to original)
  Statistics are computed on n-grams starting with v.
- **Absolute discounting** (different from original)
  The value β_v to be subtracted from all counts $N(v, w, x, y, z)$ is:

$$\beta_v = \frac{N_1(v)}{N_1(v) + 2 * N_2(v)}$$

$N_r(v)$ is # of different 5-grams starting with v and occurring exactly r times. **Notice**: if for some v the above formula is zero or undefined, we resorts to Witten-Bell method.

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* in addition compresses probabilities and back-off weights into 1 byte!

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

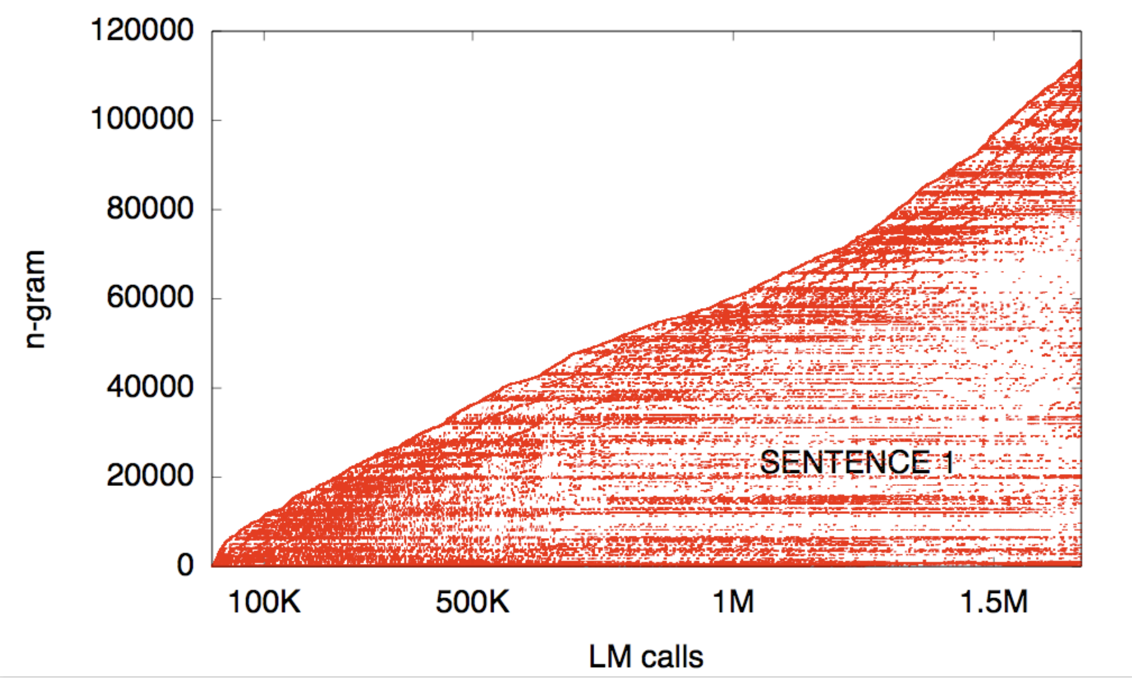
Notice: a codebook of n centers means a *quantization level* of $\log_2 n$ bits.

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
  - actual resolution of lower order n-grams is higher than that of its codebook!

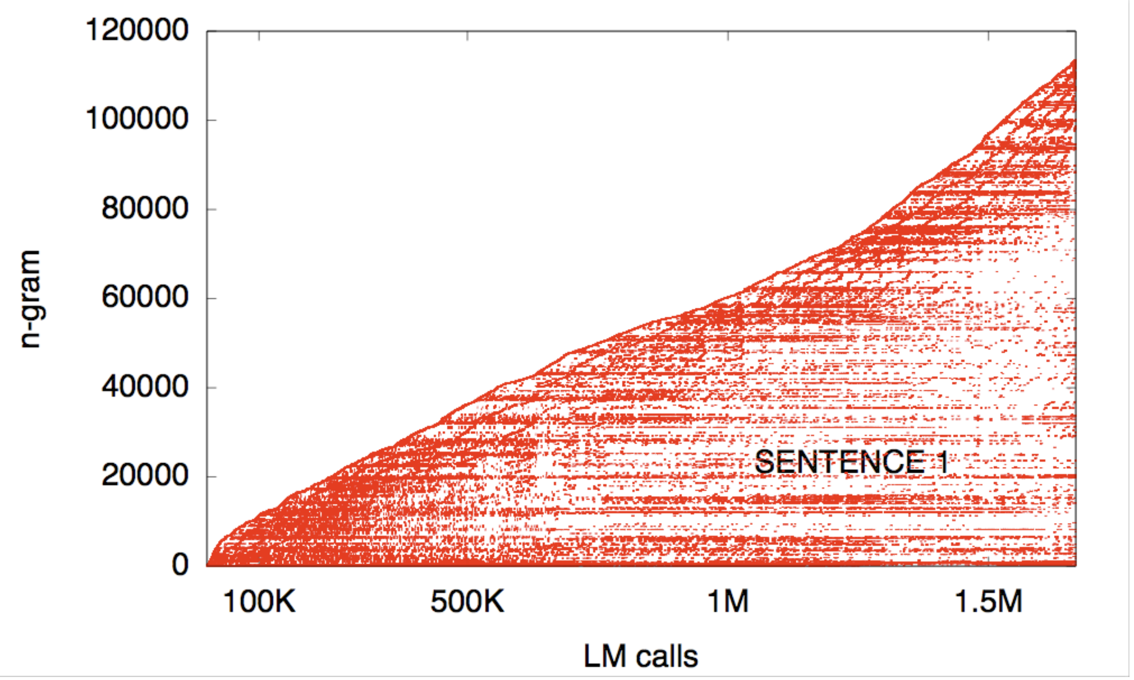
Practically no performance loss with 8 bit quantization[Federico & Bertoldi '06]

LM Accesses by SMT Search Algorithm



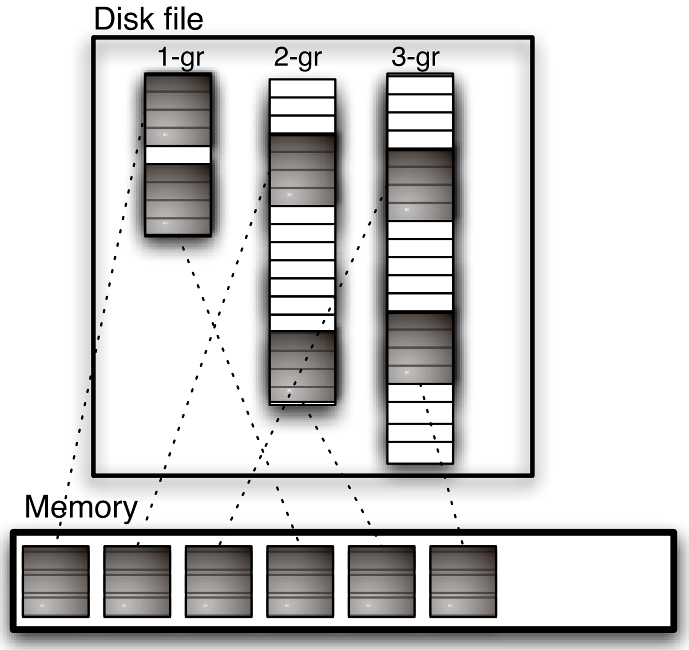
Moses's calls to a 3-gram LM while decoding into English the Europarl text:
`ich bin kein christdemokrat und glaube daher nicht an wunder . doch ich möchte dem europäischen parlament , so wie es gegenwürtig beschaffen ist , für seinen grossen beitrag zu diesen arbeiten danken.`

LM Accesses by SMT Search Algorithm



- 1.7M calls only involving 120K different 3-grams
- Decoder tends to access LM n-grams in nonuniform, *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

Experiments

Baseline: Chinese-English NIST task

- *Target Language Models*
  - 3 LMs: target part of parallel data + GigaWord + DevSets
  - 2G running words (4.5M different words)
  - 300M 5-grams (singletons pruned for GigaWord)
- *Phrase Table*
  - 90M English running words
  - 38M phrase pairs of maximum length 7
- *Monotone search*
  - permits to run fast experiments
  - you see exactly memory needed by LM
  - with lexicalized LM: +1-1.5% Bleu, +2Gb RAM, x 2.0 run-time

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

IRSTLM Library: Esperiments (NIST 2005)

LM	1gram	2gram	3gram	4gram	5gram
lrg	0.3	5.3	4.8	6.3	6.1
giga	4.5	64.4	127.5	228.8	288.6

LM	process size		caching	dec. speed (src w/s)	BLEU
	virtual	resident			
lrg SRILM	1.2Gb	1.1Gb	-	13.33	27.32
lrg	619Mb	558Mb	n	6.80	27.35
			y	7.42	

IRSTLM Library: Esperiments (NIST 2005)

LM	1gram	2gram	3gram	4gram	5gram
lrg	0.3	5.3	4.8	6.3	6.1
giga	4.5	64.4	127.5	228.8	288.6

LM	process size		caching	dec. speed (src w/s)	BLEU
	virtual	resident			
lrg SRILM	1.2Gb	1.1Gb	-	13.33	27.32
lrg	619Mb	558Mb	n	6.80	27.35
			y	7.42	
q-lrg	507Mb	445Mb	n	6.99	27.26
			y	7.52	
lrg+giga	9.9Gb	2.1Gb	n	3.52	29.15
			y	4.28	
q-lrg+q-giga	6.8Gb	2.1Gb	n	3.64	28.98
			y	4.35	

Conclusions

Efficient handling of large scale LMs for SMT:

- *Training is distributed* over many machines
  – *approximate smoothing* does not seem to hurt so far
- Run-time LM access through *compact data structure*
- While decoding one sentence *LM is loaded on-demand*
- Comparison with state-of-the-art SRILM toolkit:
  – w/o memory mapping: 60% less memory, 45% slower decoding
  – w memory mapping: *90% less memory*!
- MT system with 5-gram LM runs on 2Gb PC rather than on a 20Gb PC!

Conclusions

Efficient handling of large scale LMs for SMT:

- *Training is distributed* over many machines
  – *approximate smoothing* does not seem to hurt so far
- Run-time LM access through *compact data structure*
- While decoding one sentence *LM is loaded on-demand*
- Comparison with state-of-the-art SRILM toolkit:
  – w/o memory mapping: 60% less memory, 45% slower decoding
  – w memory mapping: *90% less memory*!
- MT system with 5-gram LM runs on 2Gb PC rather than on a 20Gb PC!

Thank You!