IRST Language Modeling Toolkit
Version 5.0
USER MANUAL

M. Federico, N. Bertoldi, M. Cettolo
FBK-irst, Trento, Italy

May 22, 2007

1 Introduction

The IRST Language Modeling Toolkit features algorithms and data structures suitable to estimate, store, and access very large LMs. Our software has been integrated into a popular open source SMT decoder called `Moses`.[1]

Acknowledgments. Users of this toolkit might cite in their publications:

> Marcello Federico and Mauro Cettolo, *Efficient Handling of N-gram Language Models for Statistical Machine Translation*, In Proc. of the Second Workshop on Statistical Machine Translation, pp. 88–95, ACL, Prague, Czech Republic, 2007.

References to introductory material on n-gram LMs are given in the appendix. Commands and scripts described in this manual are installed under the local directories `bin` and `bin/$MACHTYPE`, that we assume are included in your `PATH` environment variable. Data sets used in the examples can be found in the `example` directory.

2 Estimating Gigantic LMs

LM estimation starts with the collection of n-grams and their frequency counters. Then, smoothing parameters are estimated for each n-gram level; infrequent n-grams are possibly pruned and, finally, a LM file is created containing n-grams with probabilities and back-off weights. This procedure can be very demanding in terms of memory and time if it applied on huge corpora. We provide here a way to split LM training into smaller and independent steps, that can be easily distributed among independent processes. The procedure relies on a training scripts that makes little use of computer RAM and implements the Witten-Bell smoothing method in an exact way.
Before starting, let us create a working directory under `examples`, as many files will be created:

```
$> mkdir stat; mv train.gz stat; cd stat
```

[1] http://www.statmt.org/moses/

Step 1 First extract the dictionary from the corpus including word frequency statistics

```
$> dict -i="gunzip -c train.gz" -o=dictionary -f=y
```

Step 2 Split the dictionary into K word lists balanced according to word frequency (here, K=3):

```
$> split-dict.pl --input dictionary --output dict. --parts 3
```

which produces word lists dict.000 ... dict.002 into the subdirectory stat.

Step 3 For each word list, extract n-grams starting with words in the list:

```
$> for n in 000 001 002; do
$> ngt -i="gunzip -c train.gz" -n=3 -gooout=y -o="gzip -c > ngram.${n}.gz" -fd=
$> done
```

Step 4 For each n-gram file estimate an independent sub-LMs:

```
$> for n in 000 001 002; do
$> build-sublm.pl --size 3 --ngrams "gunzip -c ngram.${n}.gz" --sublm lm.$n --w
$> done
```

Relevant parameters:

`--size <int>` maximum n-gram size for the language model
`--ngrams <string>` input file or command to read the ngram table
`--sublm <string>` output file prefix to write the sublm statistics
`--freq-shift <int>` (optional) value to be subtracted from all frequencies
`--kneser-ney` use approximate kneser-ney smoothing
`--witten-bell` (default) use witten bell smoothing
`--prune-singletons` remove n-grams occurring once, for n=3,4,5,...
`--cross-sentence` (optional) include cross-sentence bounds

Step 5 Merge all sub-LMs into one single large file in the ARPA format (see below):

```
$> merge-sublm --size 3 --sublm lm. -lm train.lm.gz
```

In the following sections, we will talk about LM file formats, compiling your LM into a more compact and efficient binary format, and about querying your LM.

3 LM File Formats

This toolkit supports three output format of LMs. These formats have the purpose of permitting the use of LMs by external programs. External programs could in principle estimate the LM from an n-gram table before using it, but this would take much more time and memory! So the best thing to do is to first estimate the LM, and then compile it into a binary format that is more compact and that can be quickly loaded and queried by the external program.

3.1 ARPA Format

This format was introduced in DARPA ASR evaluations to exchange LMs. ARPA format is also supported by the SRI LM Toolkit. It is a text format which is rather costly in terms of memory. There is no limit to the size n of n-grams.

3.2 qARPA Format

This extends the ARPA format by including codebooks that quantize probabilities and back-off weights of each n-gram level. This format is created through the command `quantize-lm`.

3.3 Binary Formats

Both ARPA and qARPA formats can be converted into a binary format that allows for space savings on disk and a much quickers upload of the LM file. Binary versions can be created with the command `compile-lm`, that produces files with headers `blmt` or `Qblmt`.

4 LM Quantization and Compilation

A language model file in ARPA format, created with the IRST LM toolkit or with other tools, can be quantized and stored in a compact data structure, called language model table. Quantization can be performed by the command:

```
$> quantize-lm  train.lm train.qlm
```

which generates the quantized version `train.qlm` that encodes all probabilities and back-off weights in 8 bits. The output is a modified ARPA format, called qARPA.
LMs in ARPA or qARPA format can be stored in a compact binary table through the command:

```
$> compile-lm train.lm tran.blm
```

which generates the binary file `train.blm` that can be quickly loaded in memory.

5 LM Interface

LMs are useful when they can be queried through another application in order to compute perplexity scores or n-gram probabilities. IRSTLM provides two possible interfaces:

- at the command level, through `compile-lm`
- at the c++ library level, mainly through methods of the class `lmtable`

In the following, we will only focus on the command level interface. Details about the c++ library interface will be provided in a future version of this manual.

5.1 Perplexity Computation

To compute the perplexity directly from the LM on disk, we can first compile the LM in binary format, and then evaluate it on the test set with the commands:

```
$> compile-lm train.lm train.blm
$> compile-lm train.blm  --eval=test
   Nw=49984 PP=311.86 PPwp=698.96 Nbo=42747 Noov=2503 OOV=5.01
```

5.2 Probability Computations

We can compute as well log-probabilities word-by-word from standard input with the command:

```
$> compile-lm train.blm --score=yes < test

> </s>  1 p= NULL
> </s> <s>      1 p= NULL
> </s> <s> \_unk\_        1 p= -4.702528e+00 bo= 1
> <s> \_unk\_ of  1 p= -4.201473e+00 bo= 1
> \_unk\_ of the  1 p= -1.873415e+00 bo= 1
> of the senate 1 p= -1.191170e+01 bo= 1
> the senate (  1 p= -5.648314e+00 bo= 1
> senate (\_unk\_        1 p= -3.140081e+00 bo= 1
> (\_unk\_ )     1 p= -5.185384e+00 bo= 1
> \_unk\_ ) </s>  1 p= -4.405500e+00 bo= 1
> ) </s> <s>    1 p= -2.266836e+01 bo= 1
> </s> <s> 2    1 p= -9.532104e+00 bo= 1
....
....
```

the commands reports the currently observed n-gram, including `_unk_` words, a dummy constant frequency 1, the log-probability, and the information whether or no the LM performed a back-off.

A Reference Material

The following books contain basic introductions to statistical language modeling:

- *Spoken Dialogues with Computers*, by Renato DeMori, chapter 7.
- *Speech and Language Processing*, by Dan Jurafsky and Jim Martin, chapter 6.
- *Foundations of Statistical Natural Language Processing*, by C. Manning and H. Schuetze.
- *Statistical Methods for Speech Recognition*, by Frederick Jelinek.
- *Spoken Language Processing*, by Huang, Acero and Hon.

B Version History

Improvements over previous releases:

B.1 Since version 3.2

- Quantization of probabilities
- Efficient run-time data structure for LM querying
- Dismissal of MT output format

B.2 Since version 4.2

- Distinction between open source and internal Irstlm tools
- More memory efficient versions of binarization and quantization commands
- Memory mapping of run-time LM
- Scripts and data structures for the estimation and handling of gigantic LMs
- Integration of IRSTLM into Moses Decoder