# Model Building from Bitext

## 1 Introduction

Modern SMT systems typically depend on three kinds of models: a language model, a translation model (aka phrase table), and lexicalized reodering model (aka lexicalized distortion model). The latter two require a parallel corpus (aka bitext) to be built. Building those bitext models requires a number of steps:

1. Word Alignment: Here word-to-word (but not necessarily one-to-one) links are learned for each parallel sentence pair in the the bitext.
2. Word Translation Models: Here models of the form $p(w_f|w_e)$ and $p(w_e|w_f)$ are learned, where $w_e$ and $w_f$ are words.
3. Lexicalized Reordering Models: These are models of the form $p(Orientation|(p_f, p_e))$ where Orientation=monotonic,swap,discontinuous, where $p_f$ and $p_e$ are phrases.
4. Translation Models: These are models of the form $p(p_f|p_e)$ and $p(p_e|p_f)$ , where $p_f$ and $p_e$ are phrases.

With our model building tool you can do step 2 to 4 assuming that you already have a word aligned parallel corpus in hand. All these steps can be done by a single script.

`build-models-from-wordAligned-bitext.pl`

## 2 Installation

There is no need to install the model building tool itself. It works right after cloning from the repository. However, the dependencies should be addressed beforehand.

### 2.1 Dependecies

This model building tool has dependencies to three binary files from Moses translation system. These files include:

- consolidate
- extract
- score

In order to meet the dependencies, first install Moses translation tool following the installation instructions.

After installation, copy the aforementioned binary files from Moses installation directory to the following path:

```
[PATH-TO-MODEL-BUILDING-TOOL-HOME]/dependencies/moses/bin/
```

Now the tool can be used by calling it using the command given in the example below.

# 3 How to Use

A sample calling of the script is as follows:

```
./build-models-from-wordAligned-bitext.pl --input-files-prefix=aligned
--experiment-dir=./expdir  --dependencies=./dependencies --f=chinese --e=english
--a=grow-diag-final --build-distortion-model --use-dlr
--moses-orientation  --build-phrase-table >& err.log
```

- `--input-files-prefix`

  Specifies the shared prefix of the names of source, target and alignment files. This means that these three input files should have shared prefix in their names. For examples

  `aligned.chinese, aligned.english and aligned.grow-diag-final.`

- `--experiment-dir`

  Specifies the path in which the input files are placed and the final and intermediate output files of the model building process will be created.

- `--f`

  Specifies the suffix of the input source file. For example,

  `--f=chinese`

  if the source file name is like

  `[FILENAME].chinese`

- `--e`

  Specifies the suffix of the input target file. For example,

  `--e=english`

  if the target file name is like

  `[FILENAME].english`

- `--a`

  Specifies the suffix of the input alignment file. For example,

  `--a=grow-diag-final`

  if the alignment file name is like

  `[FILENAME].grow-diag-final`

- `--build-distortion-model`

  Flag to build lexicalized reordering model.

- `--use-dlr`

  Flag to generate 4 reordering orientations instead of 3 by splitting discontinuous orientation into discontinuous left and discontinuous right.

- `--build-phrase-table`

  Flag to create phrase table (translation model).

- `--moses-orientation`

  Flag to use moses style orientations to build lexicalized reordering models. The default is to use Oister style which achieves higher BLEU score comparing to moses style.

- `--dependencies`

  Specifies the path to the dependencies folder where other scripts are located.

This will take some time, depending on the size of the bitext files. If they are small ($< 2,000$ lines, for debuggin purposes), it'll take a few minutes. If they are large ($> 200,000$ lines) it can take several hours. After it has finished, you should see a number of new directories under the path to

`--experiment-dir`

. The most important one is models/model which contains the following files:

- `dm_fe_0.75.gz`

  The lexicalized reordering model.

- `lex.e2f and lex.f2e`

  The word translation models.

- `phrase-table.gz`

  The translation model.

# 4 Building Language Model

With our model building tool, you can easily build large language model from simple text. This tool provide easy inegration with the well known languge model engine SRILM.

## 4.1 Dependecies

This tool makes calls to SRILM language modelling toolkit, hence you need to install SRILM before you using this tool. Downlaod the toolkit from (http://www.speech.sri.com/projects/srilm/download.html) and follow the instruction for installtion in INSTALL (http://www.speech.sri.com/projects/srilm/docs/INSTALL) file.

## 4.2 Usage

Large language models can be built by simply running the perl script build-large-lm.pl.
A sample calling of the script is as follows:

```
./build-large-lm.pl --text=input.txt --lm=output.lm --srilm-path=/path-to-srilm-directory --order=5
```

Detailed arguments (options) are explained as follows :

- `--text`

  Specifies the input text file. Input file should contain the sentences one per line.

- `--lm`

  Specifies the name of the output file.

- `--srilm-path`

  Specifies the path to srilm top level installation directory

- `--order`

  Specifies the n-gram order of the language model requied. For example, –order=5 specifies a 5-gram language model. (default=5)

- `--working-dir`

  This tools creates multiple temporary files while building the langauge model. This option specifies the path to a directory where these temporary files should be stored. The tool deletes the file after the language model is built. (default=working-dir. If no value is provided a temporay directory is built where the script is called)

- `--keep-files`

  Specifies temporary log files not to be deleted. By default, this tool deletes temporary log files. If you need them for debugging purposes, provide this flag.

- `--smoothing`

  Smoothing or discounting techniques are used while building language model to account for sparse or rare n-grams. This tool provides two smoothing/discounting techniques. 1. Kneser-Ney smoothing : specified as –smoothing=kneser-ney or –smoothing=kneser-ney=kndiscount 2. Witten-bell smoothing : specified as –smoothing=wbdiscount or –smoothing=-wbdiscount. (default=kndiscount)

- `--no-interpolation`

  Specifies no interpolation of the discounted n-gram probability estimates with lower-order estimates. By default, this tool provides for such an interpolation (This sometimes yields better models with some smoothing methods).

- `--min-counts`

  Sets the minimal count of N-grams of order n that will be included in the LM. All N-grams with frequency lower than that will effectively be discounted to 0. For example, –min-counts=2-2-2-2-2, specifies that for a LM of order 5, the minimum frequency of all n-grams of size 5 or lesser should be 2. Any n-gram with value less that 2 will be considered non-existent. (default=1-1-1-2-2)

- `--batch_size`

  Specifies number of sentences per batch to be processed. This tool splits the input text in batches and finally combines output of each batch in a single language model. A higher batch size results in higher parallelization and hence builds the LM faster. However, it increases the memory requirement. A lower batch size requires low memory, however, splits data in less number of batches and hence results in slow speed. (default=1000000). Reduce the number of batches if you have memory limitations.

- `--pre-processing`

  Specifies different pre-processing options to be applied to the input text before building language model. For example, –pre-processing=lc specifies that text should be converted to lowercase. This tool by default provides 4 pre-processing operations :

  - `--pre-processing=lc`
    : All the letters in the text to be lowercased
  - `--pre-processing=numsub`
    : All number should be transliterated. For example '11' should be converted to 'eleven'
  - `--pre-processing=dedupl`
    : Sort and remove duplicate entries from the files.
  - `--pre-processing=sent_tags`
    : Apply tags $< s >$ and $< /s >$ at start and end of each sentence.

# 5 Language Model Interpolation

In applications of Statistical Machine Translation, at some instances, it may be required to interpolate two or more different language models into one single Language model. For example, when building Machine Translation for specific domains such as healthcare/medical, we would like to build a language model only from the training data of medical domain. However, using additional data from other domains may result in better translation outputs. In such cases, one may want to give different weightage to data or language model from different domains. For ex, in this case higher weightage for medical domain LM and lower for general or other domain data. Our tool provides support for a such an interpolation.
A sample call to the interpolation script is as follows:

```
./build-interpolated-lm.pl
--input-corpora=1-1-1-2-2:kndiscount:input1.txt,1-1-1-1-1:wbdiscount:input2.txt
--input-lms=pretrained-1.lm,pretrained-2.lm,pretrained-3.lm
--lm=interpolated.lm --ppl=dev-ref.txt
--external-path=/absolute-path-to-srilm/ --order=5
```

Detailed usage is as explained below :

- `--input-corpora`

  Specifies comma-separated list of input files format:

  `--input-corpora=<tuple1>,<tuple2>,...`

  where

  `tupleN = <min-counts>:<smoothing>:text_file`

  Here $text_file$ is a plain text input file (one sentence per line) and **min-counts** and **smoothing** are the corresponding minimum frequency count and smoothing options to be used for langauge model built from $text_file$. For example, in the example call above, we want to interpolate data from two different text files *input1.txt* and *input2.txt*. For the former we want to use a minimum frequency count of *1-1-1-2-2* and smoothing method *kndiscount* (Kneser-Ney) and for the later, we want to use a minimum frequency count of *1-1-1-1-1* and smoothing method *wbdiscount* (Witten-bell). If you do not require to build a language model form plain text, and only use pretrained langauge models, do not specify this option.

- `--input-lms`

  Specified comma-separated list of already built (pretrained) LMs. If we want to simply interpolate pre-trained LMs, a comma seperated list of the names/paths of the all pre-trained model is provided to this option. For example, in above sample call, we want to interpolate 3 language models. *pretrained-1.lm*, *pretrained-2.lm*, *pretrained-3.lm*
  . Do not specify this option, if you do not want to use any pretrained LM. **Note** : For interpolation, all the pre-trained language models should have same *order*

- `--ppl`

  Specifies name of the text file used for perplexity minimization.
  As described in the introduction of this section, in LM interpolation, we want to assign different weights to the data from different sources. These weights are calculated by minimzing perplexities of input language models on a single sample development text. Ideally, this text should have text similar to that of test domain. In the sample call above, we provide text file *dev-ref.txt* for perplexity minimization

- `--num-parallel`

  Specifies number of parallel builds (default=1). This dependes on the size of the corpora. It's recommended to keep this value ¡5.

- `--delete-builds`

  : Specifies if the individual langauge models built from plain text to be deleted or retained. (default=true)

Following options specified for output langauge model apply same as described in Section 5.

- `--lm`

  : Path to output language model file

- `--srilm_path`
- `--batch-size`
- `--pre-processing`
- `--working-dir`
- `--order`

  **Note** : Order of interpolated output language model should be same as pretrained input language models

- `--min-counts`
- `--no-interpolation`
- `--keep-files`