Word Alignment Using Giza++ And Cygwin On Windows

Ms. Ankita Sati
M.Tech Student, Banasthali Vidyapith, Banasthali, Jaipur

Abstract

Word alignment is the natural language processing task of identifying translational relationships among the words and multi-word units in parallel corpora. Automatic word alignment in bilingual or multilingual parallel corpora has been a challenging issue for natural language processing. Giza++ is the automatic word alignment tool. In Linux system environment, it is very common and convenient to use the word alignment generated from GIZA++ for most statistical machine translation systems. Many researchers conducting their research under the Windows platform .In this paper we discuss how we used Giza++ in windows and output of their results.

1. Introduction

Word alignment is the task of identifying translational relations between words in parallel corpora with the aim of re-using them in natural language processing. Typical applications that make use of word alignment techniques are machine translation and multi-lingual lexicography.
Several tools had been developed for automatic word alignment. Some of the tools are listed below:

1. Giza++
2. The Berkeley Word Aligner
3. Natura Alignment Tools (NATools)
4. UNL aligner
5. Geometric Mapping and Alignment (GMA)

There are many applications for word alignment in natural language processing, and most of them depend on the quality of word alignment (Och and Ney, 2000; Yarowsky and Wicentowski, 2000). A frequently used application system for word alignment is the automatic extraction of bilingual lexicon and terminology from parallel corpus (Smadja et al.,1996; Melamed, 2000). Och and Ney (2003) compare various methods for computing word alignment using statistical and heuristic models and then develop a statistical word alignment toolkit, GIZA++, which is the mostly used package in statistical machine translation (SMT) nowadays.

2. Giza++

GIZA++ is part of the statistical machine translation toolkit used to train IBM Model 1 to Model 5 (Brown et al., 1993) and the Hidden Markov Model (HMM) (Och et al., 2003). It is part of the SMT toolkit EGYPT which was developed by the SMT team during the summer workshop in 1999 at the Center for Language and Speech Processing at Johns-Hopkins University (CLSP/JHU) [1].

GIZA++ is extension of GIZA. Giza++ includes a lot of additional features. The extension of GIZA++ were designed and written by Franz Josef Och.

GIZA++ has following features:

- Implement full IBM-4 alignment model
- Implement IBM-5,dependency on word classes, smoothing
- Implement HMM alignment model
- Smoothing for fertility, distortion/alignment parameters
- Improved perplexity calculation for models IBM-1,IBM-2 and HMM

3. Installation on Windows

The basic aim of this paper is to show the working of Giza++ in windows. For this we have first installed the Giza++ in windows. Cygwin is used for running Giza++ in windows. Cygwin is free software that provides a Linux-like environment and software tool set to users of any modern version of MS-Windows for x86 CPUs (NT/2000/XP/Vista/7). Cygwin consists of a UNIX system call emulation library, cygwin1.dll, together with a vast set of GNU and other free software applications organized into a large number of optional packages.

3.1 Installation of Cygwin

Download the cygwin software from www.cygwin.com/setup.exe, and then double-click to

install Cygwin on your computer. Install the binaries (.bin) of the following additional packages:

- make
- g++ (currently 4.3.4)
- autoconf
- automake
- libtool
- boost
- libboost (newer than 1.31.0)
- flip (optional, but useful)

Some of these packages will also prompt you to install related or dependent packages. These additional packages are generally necessary, so just accept the defaults. Run Cygwin now for the first time, so that your home directory (/home/your username/) is created.

3.2. Installation of Giza++

For installation of Giza++ the following packages are needed:
GIZA++ package

- developed by Franz Och
- www-i6.informatik.rwth-aachen.de/Colleagues/och

mkcls package

- developed by Franz Och
- www.-i6.informatik.rwth-aachen.de/Colleagues/och

The both package is available in one tar file which can be downloaded from http://code.google.com/p/giza-pp/downloads/list.
In order to compile GIZA++, g++ compiler version 3.3 or higher is needed. Giza++ Install by issuing command *$(MAKE) -C GIZA++-v2 mkcls-v2*

After following binaries files are created

- Giza++.exe
- mkcls.exe
- snt2cooc.out
- plain2snt.out

Copy the binaries into a top-level ~/bin/ directory for easy access

Some other package is also need for lowercasing and tokenizing the sentences. For this download the **scripts.tgz** from
http://www.statmt.org/wmt07/baseline.html

and extract it in a folder name scripts. These scripts include:

- Tokenizer scripts/tokenizer.perl
- Lowercaser scripts/lowercase.per

4. Implementation Details

In this step we describe how we align the parallel corpora using Giza++.For this first we need the parallel corpora and convert into one sentence per line.Check that the corpora have the same number of lines and that they are correctly aligned. A word alignment is done between two languages. We call the two languages the source language and the target language. This is important in order to correctly do the word alignment, so decide which language will be the source and which the target. We are using source language as English and target language as Hindi. We name the source language raw.src and target language raw.trg
After this Following steps are followed:

4.1 Pre-processing

We have to clean up the corpora, set every word in lower case and separate every word from each other (or we can say "tokenizing") We are using source language as English and target language as Hindi, so tokenizing, lowercasing is done only for English
Now enter the subdirectory scripts, and take the script tokenizer.perl and the directory nonbreaking_prefix (they should be in the same directory!).
The nonbreaking prefix let the tokenizer keep together words like "Mr.". Normally the tokenizer would have broken it into two words: "Mr" and ".", but we know that the final dot is useful, not a real punctuation.
For tokenizing the source language run the following command:

tokenizer.perl -l src < raw.src > raw.tok.src

After the tokenizing run the lowercase command for source language:

lowercase.perl -l src < raw.tok.src > raw.lw.src

4.2 Create files needed for GIZA++

The files that are used in word alignment first should convert into Giza++ format. Use plain2snt.out to convert corpus into GIZA++ format. Steps are

- Run plain2snt.out located within the GIZA++ package
- ./plain2snt.out raw.lw.src raw.trg

Files created by plain2snt

- raw.lw.src.vcb
- raw.trg.vcb

- raw.trg_raw.lw.src.snt
- raw.lw.src_raw.trg.snt

raw.lw.src.vcb consists of:
- each word from the English corpus
- corresponding frequency count for each word
- an unique id for each word

raw.trg.vcb
- each word from the Hindi corpus
- corresponding frequency count for each word
- an unique id for each word

raw.tr_raw.lw.src.snt consists of:
- each sentence from the parallel English and Hindi corpus translated into the unique number for each word

4.3 Create mkcls files needed for GIZA++

After this we have to generate word classes using mkcls. This can be done by running following commands:

- Run mkcls which is not located within the GIZA++ package
- ./mkcls –praw.lw.src –Vraw.lw.src.vcb.classes
- ./mkcls –praw.trg –Vraw.trg.vcb.classes

Files created by mkcls
- raw.lw.src.vcb.classes
- raw.lw.src.vcb.classes.cats
- raw.trg.vcb.classes
- raw.trg.vcb.classes.cats

.vcb.classes files contain:
- an alphabetical list of all words (including punctuation)
- each words corresponding frequency count

.vcb.classes.cats files contain:
- a list of frequencies
- a set of words for that corresponding frequency

4.4 Run GIZA++

After the creating word classes we run GIZA++ located within the GIZA++ package by issuing following command :

./GIZA++ -S raw.lw.src.vcb –T raw.trg.vcb –C raw.lw.src_raw.trg.snt

After running this command various file created, main word alignment is in actual.ti.final

- Decoder.config
- ti.final
- actual.ti.final
- perp
- trn.src.vcb
- trn.trg.vcb
- tst.src.vcb
- tst.trg.vcb
- D4.final
- a3.final
- A3.final
- t3.final
- d3.final
- d4.final
- n3.final
- p0_3.final
- gizacfg

Decoder.config
- file used with the ISI Rewrite Decoder
- developed by Daniel Marcu and Ulrich Germann

trn.src.vcb
- list of English words with their unique id and frequency counts
- similar to raw.lw.src.vcb

trn.trg.vcb
- list of Hindi words with their unique id and frequency counts
- similar to raw.trg.vcb

tst.src.vcb
- blank

tst.trg.vcb
- blank

ti.final
- file contains word alignments from the English and Hindi corpi
- word alignments are in the specific words unique id
- the probability of that alignment is given after each set of numbers
  Ex:
  3 0 0.237882
  1171 1227 0.963072

actual.ti.final
- file contains word alignments from the English and Hindi corpora
- words alignments are the actual words not their unique id’s
- the probability of that is alignment is given after each set of words
  Ex: of NULL 0.237882
  याचिकाओंaffidavits 0.25

perp
- list of perplexity for each iteration and model
  #trnsz tstsz -iter model trn-pp test-pp trn-vit-pp tst-vit-pp 19 0 0 Model1 204.141 N/A 5098.14 N/A
  trns – training size

tstsz – test size
iter – iteration
trn-pp – training perplexity
tst-pp – test perplexity
trn-vit-pp – training viterbi perplexity
tst-vit-pp – test viterbi perplexity

a3.final

- contains a table with the following format:
  - i j l m p (i / j, l, m)
  - j = position of target sentence
  - i = position of source sentence
  - l = length of the source sentence
  - m = length of the target sentence
  - p(i / j, l, m) = is the probability that a source word in position i is moved to position j in a pair of sentences of length l and m

Ex:
0 1 1 60 5.262135e-06
0 – indicates position of target sentence
1 – indicates position of source sentence
1 – indicates length of source sentence
60 indicates length of target sentence
5.262135e-06 – is the probability that a source word in position 1 is moved position 0 of sentences of length 1 and 60

d3.final

- similar to a3.final with positions i and j switched

n3.final

- contains the probability of the each source token having zero fertility, one fertility, … N fertility

t3.final

- table after all iterations of Model 4 training

d4.final

- translation table for Model 4

D4.final

- distortion table for IBM-4

gizacfg

- contains parameter settings that were used in this training.
- training can be duplicated exactly

p_03.final

- probability of inserting null after a source word
- file contains 0.883714

5. Experimental Results

For a word alignment we use English–Hindi parallel corpora which contain 500 sentences. After running these sentences in Giza++, Accuracy of output is 50% to 60 %. Comparing the results in Linux environment the output is nearly same.

GIZA++ allows aligning one token from the source language to multiple tokens in the target language, i.e. one-to-many alignments, but does not allow multiple tokens from the source language to align to the same target token. Due to this asymmetry, running GIZA++ with source and target languages swapped produces different alignments. Because we desire a high level of precision for prealignments, we run GIZA++ twice, alternating the order of source and target languages, then take the fine intersection of the resulting alignments. The intersection necessarily contains only one-to-one alignments due to the restrictions of the GIZA++ structures.

A variant of GIZA++ is MGIZA++, a derivative of GIZA++ which allows users to save trained model states.

6. Conclusion

The use of word alignment in natural language processing has increased dramatically in recent years, especially in the development of statistical machine translation. Manual word alignment can be an expensive, time consuming process, especially given the data volumes produced at organizations such as the Linguistic Data Consortium In this paper, we adapted the word alignment model GIZA++ to the work on Windows environment. The contribution of this work is to find a way in making GIZA++ run under the Windows environment.

10. References

[1] F.J.Och, “GIZA++: Training of statistical translation models”, [Online]. Available at: http://fjoch.com/GIZA++.html
[2] http://wiki.apertium.org/wiki/Using_GIZA%2B
[3] Giza++software [Online]. Available: http://code.google.com/p/giza-pp/downloads/list
[4] “Statistical Machine Translation” System User manual and Code Guide {online].Available: http://www.statmt.org/moses/manual/manual.pdf/