Decoding in Statistical Machine Translation Using Moses And Cygwin on Windows

Ms. Pragati Vaidya
M.Tech Student, Banasthali Vidyapith, Banasthali, Jaipur

***Abstract*— Decoding is an integral part in SMT most essential components in decoding are Language modeling and reordering. The Moses toolkit generally used for decoding, in which SRILM, and GIZA++ are the automatic tools, which is used for probabilistic language modeling and reordering respectively. The Aim of this paper is to explore the SMT using the Moses toolkit for English-Hindi language pair. In Linux environment these tools are very habitual and easy to use but some researchers conducted their research on windows. Whenever installed these tools in both environments, then measure the result and the time difference. In this research paper we discuss how we used Moses, Srilm and Giza++ in windows and their results.**

***Keywords*— *Decoder, Language model, Alignment model, Moses, Srilm, Giza++*.**

I. INTRODUCTION

the main problem arise in SMT system that is selecting the best translated target sentences which efficiently search for the source sentence that is completely satisfied. The words are chosen which has extreme possibility of being a translated translation. According to Bayes rule

Pr(S/T) = argmax P (T) P(S/T)

Where S denote the source language and T denote the target language. Lots of tools which had been developed for decoding in Statistical Machine Translation. Some of the tools are given below:

1. Moses
2. Marie Decoder
3. Joshua Decoder
4. Phramer Decoder
5. GREAT Decoder

Moses is a collection of tools in SMT for decoding. A decoder is a single C++ application in which a trained Machine translation system translates the source sentences into its equivalent target sentence. In this research paper we consider English-Hindi language pair. Language modelling is a task of estimating the probability of each unit text, phrases, sentences, paragraphs. Reordering is a Natural language Processing task to identify the correct translation of each other based information found on parallel text.

SRILM and GIZA++ are the automatic tools, which is respectively used for probabilistic language modelling, and reordering.

A. Hardware Requirement

The hardware requirements for the SMT are:
High-end server with the minimum 2 GB RAM
Windows

II. MOSES

Moses is an open source project, Licensed under the LGPL.[1] Moses is an collection of tools in SMT, which is written in Perl with some in C++. It takes the raw data or parallel or monolingual, bilingual data to perform the translation process. In 2005 Hieu Hoang student of Philipp Koehn starts Moses as successor to Pharoah.

The job of the Moses decoder is to find the highest scoring sentence in the target language (according to the translation model) corresponding to a given source sentence. **[1]**

III. INSTALLATION ON WINDOWS

The elementary resolution of this paper is to demonstrate the working of Moses, Giza++, and SRILM in windows. For this we have to first install Moses, Giza++, and SRILM in windows. Cygwin is used for running Moses, Giza++, and SRILM in windows. Cygwin is a freeware software, It consist of two parts A Dynamic Library DLL (Cygwin1.dll) which acts as a Linux API emulation layer Providing substantial Linux API functionality and A collection of tools which provide Linux like environment, and it also deliver some software tools which is set to Users to any latest version of MS-Windows for x86 CPUs (NT/2000/XP/Vista/7/8).

3.1 Installation of Cygwin:

Download the software of cygwin go to this link www.cygwin.com/setup.exe, (~250 Kbytes). Then double click to install the Cygwin on your computer. To install the Cygwin It is necessary to install all binaries (.bin) files following 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 are automatically installed from the internet but some of these packages are dependent and request you to install or not. These additional packages are frequently essential for cygwin, so just accept the defaults. Now run Cygwin for the first time, So Select your home directory (/home/your username/) is created.

3.2 Installation of Moses:

For the compilation of Moses, Moses uses bjam (boost build system). For the installation of Moses basic models are needed: such as Language Model and Alignment Model with the help of these models we effortlessly decode the sentences from one language to corresponding to another.

```
cd ~/mosesdecoder
./bjam –help
```

3.2.1 Alignment Model:

Alignment (Reordering) is a Natural language Processing task to detect the correct translation of each other based statistics found on parallel text. Alignment is dominant approach for decoding and accuracy of the translated sentence is depending on the word alignment. Find the almost certainly translations of an SL word, irrespective of position.
Several tools that had been developed for automatic alignment some of these are given below:

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

There are many functions of alignment in SMT

- Computing the length of the source language sentences according to the length of the target language sentence.
- Determine the position in target sentence alignment aligned to the first word of source language sentence.

3.2.1.1 Giza++:

Giza++ is a part of statistical machine translation toolkit (http://www.clsp.jhu.edu/ws99/projects/mt/toolkit/) which is used to train the IBM Model 1 to Model 5 (Brown et al., 1993) and use the Hidden Markov Model (HMM) (Och et al., 2003)[2] and uses these models to compute Viterbi Alignments for statistical machine translation. Giza++ is an implementation of IBM model and it treats word alignment as a hidden process. Giza++ is a tool which is developed by Franz Josef Och and is an extension of the program GIZA which was developed by the Statistical Machine Translation team during the summer workshop in 1999 at the Center for Language and Speech Processing at Johns-Hopkins University (CLSP/JHU). Giza++ is an extension of Giza++.

3.2.1.1.1 Features of Giza++ not in Giza:

- Improved perplexity calculation for models IBM-1, IBM-2 and HMM (the parameter of the Poisson-distribution of the sentence lengths is computed automatically from the used training corpus)
- Implementation of a variant of the IBM-3 and IBM-4 (-deficient Distortion Model 1) models which allow the training of
    - Smoothing for fertility, distortion/alignment parameters
    - Significant more efficient training of the fertility models
- Implements IBM-5: dependency on word classes, smoothing.
- Implements HMM alignment model.
- Correct implementation of pegging, implemented a series of heuristics in order to make pegging sufficiently efficient

3.2.1.1.2 Packages used in Giza++

- 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

3.2.1.2 Installation of Giza++:

Download the latest version of Giza++ from internet and as well as extract these file from cygwin using commands

```
wget http://giza-pp.googlecode.com/files/giza-pp-v1.0.7.tar.gz
tar xzvf giza-pp-v1.0.7.tar.gz
cd giza-pp
make
```

3.2.1.2.1 Compilation:

To compile GIZA++:

1. The GNU compiler 2.96 is needed.
2. Two changes need to be made to the Makefile:
    - Set the path where GIZA++ will be installed. For example:

      NSTALLDIR = /home/bthomson/GIZA++
    - Set the location of the gnu compiler.
        - For example: CC = g++
3. Type 'make' to make the executable program

To compile mkcls:

1. Three changes need to be made to the Makefile:
   - Set the path where mkcls will be installed. For example: INSTALLDIR = /home/bthomson/mkcls
   - Set the shell that you are using.
     - For example: SHELL = bash
   - Set the location of the gnu compiler.
     - For example: CC = g++
2. Type 'make' to make the executable program.

To compile plain2snt.cc:

./g++ -o plain2snt.out plain2snt.cpp
Giza++ Install by issuing command
$(MAKE) -C GIZA++-v2 mkcls-v2
This should create the binaries
`~/giza-pp/GIZA++-v2/GIZA++`,
`~/giza-pp/mkcls-v2/mkcls`
`~/giza-pp/GIZA++-v2/snt2cooc.out`

These required to be copied to ~/bin/ directory for easy access and Moses can find them, some other package is also required 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

3.2.1.2.2 Pre-processing:

The performance of the statistical machine translation is directly depends upon the quality of the corpus. Pre-processing of corpus we have to perform tokenization and cleaning are the essential parts to build the highly accurate corpus. Tokenization means split the sentence in chunks and cleaning means to remove the long sentences, this can cause problems with the training process and obviously mis-aligned sentences.

Tokenizer can be run as follows:

~/mosesdecoder/scripts/Tokenizer/tokenizer.perl -l hi \
< ~/corpus/training/en-hi.hi \
> ~/corpus/en-hi.tok.hi
~/mosesdecoder/scripts/Tokenizer/tokenizer.perl -l en \
< ~/corpus/training/ en-hi.en\
> ~/corpus/ en-hi.tok.fr

Truecasing run as follows:

~/mosesdecoder/scripts/recaser/truecase.perl \
--model ~/corpus/truecase-model.hi \
< ~/corpus/en-hi.tok.hi \
> ~/corpus/en-hi.true.hi
~/mosesdecoder/scripts/recaser/truecase.perl \
--model ~/corpus/truecase-model.en \
< ~/corpus/ en-hi.tok.en \
> ~/corpus/ en-hi.true.en

Cleaner can be run as follows:

Finally we clean, limiting the length of sentences, 50
~/mosesdecoder/scripts/training/clean-corpus-n.perl \
~/corpus/en-hi.true en hi \
~/corpus/en-hi.clean 1 50

Training Giza++ for English-Hindi word alignment

1. To compile a bilingual corpus which is sentence aligned, create two files from cygwin For example:
   cat > english
   cat > hindi

2. Run plain2snt.out which is located in the GIZA++ package. The first argument is the source language and the second argument is the target language.
   ./plain2snt.out english hindi

Three output files will be created:

- english.vcb: Contains each word from english corpus and corresponding frequency count and a unique ID
- hindi.vcb: Contains each word from hindi corpus and corresponding frequency count and a unique id.
- englishhindi.snt: Each sentence from parallel hindi and english corpus translated into a unique number for each word.
- Copy these files in mkcls-v2.

3. Run mkcls to create word classes. mkcls is a separate package.
   ./_mkcls -penglish -Venglish.vcb.classes
   ./_mkcls -phindi -Vhindi.vcb.classes

Four output files will be created:

- english.vcb.classes
- english.vcb.classes.cats
- hindi.vcb.classes
- hindi.vcb.classes.cats

.vcb.classes file: Contains an alphabetical list of all words and corresponding frequency count.
.vcb.classes.cats file: Contains a list of frequencies and a set of words for that corresponding frequency

3.2.1.3 Run Giza++:

To generate the alignment between english to hindi sentences
Command:
./GIZA++ -T english.vcb -S hindi.vcb -C englishhindi.snt

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

1. Translation table (*.t*.*)
   - The format of the translation table is
     Source_id target_id P(target_id|source_id)

2. Fertility table (*.n3.*)

- source token id p0 p1 p2 . . . pn
- Where p0 is the probability that the source token has zero fertility; p1, fertility one, ..,and n is the maximum possible fertility as defined in the program.

3. Probability of inserting a null after a source word (*.p0*)
- Contain only one line with the probability of not inserting a NULL token.

4. Alignment tables (*.a*.*)

- The format of each line is as follows:
- 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

5. Distortion table (*.d3.*)
- The format is similar to the alignment tables but the position of i and j are switched:
- j i l m P(j | i, l, m)

6. Distortion table for IBM-4 (*.d4.*)
7. Distortion table for IBM-5 (*.d5.*)
8. Alignment probability table for HMM alignment mode (*.A3.*)
9. Perplexity File (*.perp)
10. Revised vocabulary files (*.src.vcb, *.trg.vcb)
11. Final parameter file: (*.gizacfg)
12. Final result: 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

For word alignment we used English-Hindi parallel corpus which contain 1500 sentences. After running these sentences the output is nearly same in both environment and the accuracy of output is nearly 50% to 60%.Giza++ allows one to many alignment that means one token from the source token is aligned to many tokens from the target sentence. But it's not allow many to one word alignment that means multiple tokens from the source sentence is aligned to one token from the target sentence.

3.2.2 Language Model:

Language modelling is a task of estimating the probability of each unit text, phrases, sentences, paragraphs. LM is input to any decoder system and it also one of the most time consuming step in decoder.

P (e) = P(w1w2…wn)

=P(w1)P(w2/w1)P(w3/w1w2)
=P(wn/w1w2w3…..wn-1)

Several tools that had been developed for LM for Moses some of these are listed below:
1. SRILM
2. IRSTLM
3. KenLM

In this research paper we take SRILM to run Moses decoder.

3.2.2.1 SRILM:

SRILM support Moses SMT system and Soshua hierarchical phrase based SMT system. SRILM 1.5.6 is the current and stable version.

http://www.speech.sri.com/projects/srilm/download.html
With this link we download the latest version of SRILM after registration.

3.2.2.2 Installation of SRILM:

Download the Latest version of SRILM Toolkit and then install the SRILM into the Cygwin environment.

1. Create the srilm directory if it doesn't exist
2. Extract the srilm.tgz (src files) or "srilm.zip" (executable Files)

Commands in cygwin
- $ cd /
- $ mkdir srilm //create the "srilm" directory
- $cd srilm
- $ tar zxvf srilm.tgz //extract srilm.tgz

3. Edit "c:\cygwin\home\yourname\.bashrc"
Add the following several lines into this file
- export SRILM=/srilm
- export MACHINE_TYPE=cygwin
- export PATH=$PATH:$pwd:$SRILM/bin/cygwin
- export MANPATH=$MANPATH:$SRILM/man

4. Restart "Cygwin"
5. Run cygwin
- Switch current directory to "/srilm"
- Modify "/srilm/Makefile"
- Add a line: "SRILM = /srilm" into this file
- Execute the following commands
  $ make World
  $ make all
  $ make cleanest

3.2.2.3 Run SRILM:

To Generating the N-gram Count File
Command:
- vi egg.count
- ngram-count –vocab lex.txt –text eg.train –order 3 –write egg.count – unk

Then Parameter Settings
- -vocab: lexicon file name
- -text: training corpus name

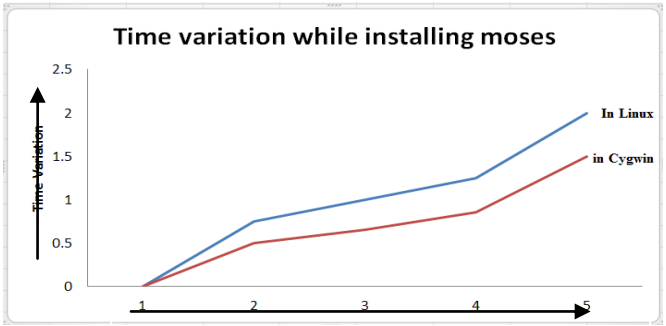
- -order: n-gram count
- -write: output countfile name
- -unk: mark OOV as <unk>

IV. RUN MOSES:

Command:
~/mosesdecoder/bin/moses-e ~/working/mert-work/moses.ini
After compiling Moses it internally run Giza++ and then SRILM and then by using the above command we run Moses on cygwin and then it gives the corresponding result.

V. EXPERIMENTAL RESULTS

For decoding process we use English-Hindi parallel corpora which contain 1500 sentences as already discussed above. After running theses sentences in Moses, the output is nearly same to both in Linux and Cygwin but when we install Moses toolkit, Giza++ and SRILM sequentially in Linux environment then decoder takes much more time to decode the source sentence, and installation time compared with the cygwin. Time measurement is also a very commanding part to installation and running of any tools. We install Moses decoder, Giza++, SRILM, twice, and in both times, we measure time variation.
To speed up the decoding process we can binaries the phrase-table and lexicalized reordering models.[1]



Sequential Installation of Moses, Giza++, SRILM

Time is precious, while we installed these tool then we analyse time variation, the above graph indicate the time variation while we install Moses, Giza++, and SRILM sequentially, Blue Line indicate the time while we install Moses, Giza++, and SRILM sequentially in Linux environment and red line indicate the time while installed these tools in Cygwin environment.

VI. CONCLUSION:

We have presented the complete installation and decoding process of Moses decoder, Giza++ (Word Alignment) and SRILM (Language Modelling), on windows, in which we only consider English-Hindi language pair, but we can also use for multilingual language pair. The contribution of this experiment is to give a way to run Moses, Giza++ and SRILM under the windows environment and decode the sentences in a little while.

REFERENCES:

[1] "Statistical Machine Translation" System User manual and Code Guide {online].Available: http://www.statmt.org/moses/manual/manual.pdf/
[2] "Moses Statistical Machine Translation System" [Online]. Available at: http://www.statmt.org/moses/?n=Moses.Baseline
[3] Giza++software [Online]. Available: http://code.google.com/p/giza-pp/downloads/list
[4] SRILM software [Online]. Available: http://www.speech.sri.com/projects/srilm/download.html
[5] Ye-Yi Wang and Alex Waibel, "Decoding Algorithm in Statistical Machine Translation".