Natural Language Processing: Using Machine Translation in Creation of a German-English Translator

Jason Ji

May 19, 2005

Abstract

The field of machine translation - using computers to provide translations between human languages - has been around for decades. And the dream of an ideal machine providing a perfect translation between languages has been around still longer. This project attempts to take the beginning steps towards that goal, creating a translator program that operates within an extremely limited scope to translate between English and German. There are several different strategies to machine translation, and this project will look into them - but the strategy taken to this project will be the researcher's own, with the general guideline of "thinking as a human." For if humans can translate between language, there must be something to how we do it, and hopefully that something - that thought process, hopefully - can be transferred to the machine and provide quality translations.

1 Background

There are several methods of varying difficulty and success to machine translation. The best method to use depends on what sort of system is being created. A **bilingual** system translates between one pair of languages; a **multilingual** system translates between more than two systems.

The easiest translation method to code, yet probably least successful, is known as the **direct approach**. The direct approach does what it sounds like

it does - takes the input language (known as the "source language"), performs **morphological analysis** - whereby words are broken down and analyzed for things such as prefixes and past tense endings, performs a bilingual dictionary look-up to determine the words' meanings in the target language, performs a local reordering to fit the grammar structure of the target language, and produces the target language output. The problem with this approach is that it is essentially a word-for-word translation with some reordering, resulting often in mistranslations and incorrect grammar structures. Furthermore, when creating a multilingual system, the direct approach would require several different translation algorithms - one or two for each language pair.

The **indirect approach** involves some sort of intermediate representation of the source language before translating into the target language. In this way, linguistic analysis of the source language can be performed on the intermediate representation. The two main variants of the indirect approach are **interlingua** and **transfer**. The interlingua approach involves translating the source language into an intermediate language or representation that is not language dependent, and then translating into the target language without "looking back" at the source. Translating to the intermediary also enables semantic analysis, as the source language input can be more carefully to detect idioms, etc, which can be stored in the intermediary and then appropriately used to translate into the target language. The transfer method is similar, except that the transfer is language dependent - that is to say, the French-English intermediary transfer would be different from the English-

German transfer. An interlingua intermediary can be used for multilingual systems.

2 Theory

Humans fluent in two or more languages are at the moment better translators than the best machine translators in the world. Indeed, a person with three years of experience in learning a second language will already be a better translator than the best machine translators in the world as well. Yet for humans and machines alike, translation is a process, a series of steps that must be followed in order to produce a successful translation. It is interesting to note, however, that the various methods of translation for machines - the various processes - become less and less like the process for humans as they become more complicated. Furthermore, it was interesting to notice that as the method of machine translation becomes more complicated, the results are sometimes less accurate than the results of simpler methods that better model the human rationale for translation. Therefore, the theory is, an algorithm that attempts to model the human translation process would be more successful than other, more complicated methods currently in development today.

This theory is not entirely plausible for full-scale translators because of the sheer magnitude of data that would be required. Humans are better transla-

tors than computers in part because they have the ability to perform semantic analysis, because they have the necessary semantic information to be able to, for example, determine the difference in a word's definition based on its usage in context. Creating a translator with a limited-scope of vocabulary would require less data, leaving more room for semantic information to be stored along with definitions. A limited-scope translator may seem unuseful at first glance, but even humans fluent in any language, including their native language, don't know the entire vocabulary of the language. A language has hundreds of thousands of words, and no human knows even half of them all. A computer with a vocabulary of commonly used words that most people know, along with information to avoid semantic problems, would therefore be still useful for nonprofessional work.

3 Development

On the most superficial level, a translator is more user-friendly for an average person if it is GUI-based, rather than simply text-based. This part of the development is finished. The program presents a GUI for the user. A JFrame opens up with two text areas and a translate button. The text areas are labeled "English" and "German". The input text is typed into the English window, the "Translate" button is clicked, and the translator, once finished, outputs the translated text into the German text area. Although

typing into the German text area is possible, the text in the German text area does not affect the translator process.

The first problem to deal with in creating a machine translator is to be able to recognize the words that are inputted into the system. A sentence or multiple sentences are input into the translator, and a string consisting of that entire sentence (or sentences) is passed to the translate() function. The system loops through the string, finding all space (' ') characters and punctuation characters (comma, period, etc) and records their positions. (It is important to note the position of each punctuation mark, as well as what kind of a punctuation mark it is, because the existence and position of punctuation marks alter the meaning of a sentence.) The number of words in the sentence is determined to be the number of spaces plus one. By recording the position of each space, the string can then be broken up into the words. The start position of each word is the position of each space, plus one, and the end position is the position of the next space. This means that punctuation at the end of any given word is placed into the String with that word, but this is not a problem: the location of each punctuation mark is already recorded, and the dictionary look-up of each word will first check to ensure that the last character of each word is a letter; if not, it will simply disregard the last character.

The next problem is the biggest problem of all, the problem of actual translation itself. Here there is no code yet written, but development of pseudocode has begun already. As previously mentioned, translation is a process. In

order to write a translator program that follows the human translation process, the human process must first be recognized and broken down into programmable steps. This is no easy task. Humans with five years of experience in learning a language may already translate any given text quickly enough, save time to look up unfamiliar words, that the process goes by too quickly to fully take note of.

The basic process is not entirely determined yet, but there is some progress on it. The process to determine the process has been as followed: given a random sentence to translate, the sentence is first translated by a human, then the process is noted. Each sentence given has ever-increasing difficulty to translate. For example: the sentence, "**I ate an apple**," may be translated via the following process:

1) Find the subject and the verb. (I; ate)

2) Determine the tense and form of the verb. (ate = past, imperfekt form)

a) Translate subject and verb. (Ich; ass) (note - "ass" is a real German verb form.)

3) Determine what the verb requires. (ate -¿ eat; requires a direct object)

4) Find what the verb requires in the sentence. (direct object comes after verb and article; apple)

5) Translate the article and the direct object. (ein; Apfel)

6) Consider the gender of the direct object, change article if necessary. (der Apfel; ein -¿ einen)

Ich ass einen Apfel.

Two separate look-ups are performed in the process of translation upon the input of the source text and the breaking of it into each separate word. The first look-up, a method listLookUp(), involves searching for a given word in a list that maps words to part of speech, a list "list.txt". ("Apple" must be found in list.txt first as a noun.) An input stream is set up using a BufferedReader wrapped around a FileReader, and directed towards list.txt. First of all, the word must be transferred into all lower case letters and boiled down into its base form in order to be found in the list. The list.txt does not contain "apples", nor does it contain "jumps" or "jumped", and does not contain "Apple" either. (It does, however, contain "ate" and "saw", as these are irregular forms.) Using the String class toLowerCase() method, "Apples" may be transferred into "apples". Using String's substring() method, "apples" may be chopped down to "apple" by creating a substring from 0 to word.length()-1. Indeed, every word is checked for an ending of -s or -ed (-ing present active tense does not occur in German, and thus is not allowed as an English input) when searching the dictionary. Using three if-statements, the word is first checked for in the dictionary, and if not found, a substring of word missing the last letter is checked for, and then finally a substring of word missing the last two letters is searched for. If word is "apple", then the first check will yield a match; if the word is "jumped", then neither the first check ("jumped") nor the second check ("jumpe") will yield a match, but the third check ("jump") will yield a match.

Having determined the part of speech which matches the word, the transla-

tion of the word must be known. In order to implement this grammar-based translation method, not only must the translation be known, but the semantic information of the word must also be known. This is the second look-up for a given word. Several text files are set up, each containing a certain type of word: nouns.txt, verbs.txt, paverbs.txt ("past tense verbs", or irregular forms), etc. Each of these text files contain a list of lines of semantic information. verbs.txt has a format of

verb - translation - past tense conjugation pattern - case (if no p)-separable prefix ("no"=none)[-conjugation]

and one line within that file is

do-machen-machte-gemacht-a-no

In this case, "do" is the verb, "machen" is its translation, and the rest of the line contains semantic information about the verb, such as its past tense forms, the case it takes, etc.

The part of speech determined from the look-up in list.txt is the name of

the specific text file which must next be searched; appending ".txt" to the end of the part of speech yields the filename of the next file to be searched. The BufferedReader is redirected to that filename, and is passed to another method, specificLookUp(). specificLookUp() takes another argument, a two-celled String array, the first cell of which is the wantedWord and the second cell of which is the ending it has ("none" if it has no ending). The ending of the word is important in English, and passing this in an array ensures that the information is passed along. specificLookUp() will check the correct list - which the BufferedReader is already open to - by reading in each line and checking a substring of the line from 0 to the first position of a dash. In the aforementioned line example, the substring would be from [0, 2), including character 0 and 1, or "do". Having passed the ending information along with the word into the method, only one check is required instead of three, the one corresponding to the correct ending information of the word. For example, if the information passed was "jumps" and "s", then the check performed would be a substring from 0 to word.length()-1; if the information passed was "jump" and "none", then the check performed would be the whole word. Once the word is found, the entire line is read in and one word is appended to the end of the line, to increase the semantic information. For example, If the word is a verb and has an 's' ending, then "-3rd" is appended, because this information clearly means the verb is in third-person form.

4 References

(I'll put these in proper bibliomumbojumbographical order later!)

1. http://dict.leo.org (dictionary)

2. "An Introduction To Machine Translation" (available online at http://ourworld.compuserve.com/homepages/WJHutchins/IntroMT-TOC.htm)

3. http://www.comp.leeds.ac.uk/ugadmit/cogsci/spchlan/machtran.htm (some info on machine translation)

4. http://dict.tu-chemnitz.de/ (dictionary)