Accessible Hybrid-Statistical Machine Translation for Small-Sized Corpora

Submitted to Siemens Competition in September 2015

Abstract

Since their introduction in the late 1990's by IBM's research labs and with growing computational power in the cloud, Statistical Machine Translation (SMT) algorithms have become state-of-the-art parts of our everyday Internet-using lives—but only for the modern languages that have large parallel text bases to train SMT algorithms. But what about dying or dead languages, which might only have 1,000 or less sentences in their entire lexicons? Traditional SMT techniques, while cheap and easier than a rule-based approach which would require language experts to manually create rules for every complex aspect of translation, fail to produce reliable results at less than 10,000 sentence pairs. In this paper, I explore a new hybrid-based method that includes text markers and elements of IBM's original Models 1-5 that produce reliable translation of Anglo-Saxon, or Old English, from Modern English, using a pair of corpora of only 1,000 words.

Introduction

The power of language is incredibly evident throughout human history. The unearthing of the Rosetta Stone, literally an old rock with a few hundred words written on it, provides modern analysts with nearly everything they know about ancient Egyptian hieroglyphics. Wars have been fought over the cultural divides of language, and linguistics have played a role in the most memorable conflicts: Alan Turing's attempts at Nazi code-breaking provided the foundations for modern computer science. His so-called "enigma machine" demonstrate the power of understanding language as well as harnessing linguistics with computational technology.

More broadly, language plays a significant role in nationalism and common identity. It is for this reason that I embarked on my quest of researching machine translation methods in order to derive an algorithm that could satisfactorily translate text from English into a given target language. The algorithm can also, of course, be re-purposed for the opposite effect, but `English -> otherLanguage` is generally more difficult given the analytical structure of English, which almost entirely lacks inflections or cases that a translator will need to be able to grammatically infer for other languages.

Statistical Machine Translation, or SMT, is a technique pioneered by researchers at IBM's labs in the 1990's and which took the linguistics world by storm as soon as computers were fast enough and had enough memory to handle the new translation process. Surprisingly, however, the field has changed very gradually over the past 10 years, and the models IBM invented in 1993 are still of practical use to an independent researcher in a position like my own [without the expensive and proprietary technologies of Microsoft, Google, etc.].

I chose to use modern standard German as my target language for a few practical reasons. A majority of the literature on machine translation has target examples in French, but

a) I don't speak French and b) French is fairly similar to English in syntax and its analytical style I needed a more suitable challenge for my model: inflections and cases. My ultimate goal is to provide a translation framework capable of turning modern English into ancient or dying languages, and the lack of a full-sentence translation for Anglo-Saxon, or Old English, piqued my attention for this project.

The difficulty of transliterating Old English from Middle English, and the reason that my algorithm is needed, is because of the limited text available in Old English, not to mention well-translated parallel texts in English, the desired source language. Combined, the current known legible texts written in Anglo-Saxon total at most 3,000 sentence pairs but probably are closer to a useful 2,000 in practice, well below the requisite 10,000 for baseline SMT using modern technologies. These 10,000 to 100,000+ pairs are easily found in the transcripts of the United Nations proceedings, which are parallel sentence-by-sentence, accurate, and accessible not to mention numbering in the millions over the past few decades. But alas, Bolo and ancient Anglo-Saxon are nowhere to be found here, so it is imperative that an algorithm for SMT, one which doesn't require the time-consuming efforts of a rule-based approach, is able to work for even struggling global languages.

Methods

Thanks to the cloud computing revolution, gaining cheap and fast access to a huge amount of computational power is possible. I used a multi-threaded virtual machine with 16 virtualized processing cores for the majority of the calculations in my test programs, with the training data broken up into separate CPU cores for processing as separate `Process()` modules. This enabled me to take full advantage of the powerful system at my disposal, which is cost-effective at about $2 per hour, all over the Internet and instantly available and configurable. In each thread, I used a `for` statement in Python to break the model's neces-

sary probability calculations into chunks, most of which is done sentence-by-sentence, then corroborated in a single thread. The sentence-by-sentence processing can be efficiently divided rather than waiting on one core. Availability of sufficient quantities of memory is also critical: reading, for a large corpus, 50,000 sentence pairs, averaging 27 words each, is extremely memory intensive (I suppose it's at my advantage that I used only 1,000 sentence pairs!) In Unicode, with German including 32-bit, or 4-byte, characters every once in a while (for example, ü and ß), and Anglo-Saxon containing rarer tokens like thorn, that equals, at about 6 characters per word:

$$\begin{aligned} &50{,}000 \text{ sentences} \\ &\times 2 \text{ sets (parallel corpora))} \\ \times 27 \text{ words } &^{4 \text{ (probabilities of matching phrases in each sentence pair)}} \\ &\times 1.5 \text{ bytes} \\ &\approx 80 \text{ gigabytes} \end{aligned}$$

In order to decrease latency, a full implementation of this algorithm would require a huge amount of memory, and every time a model was to be used, the translator would have to load the entire set of values into memory and search it efficiently. I won't be covering optimization, but rather a simpler approach in the implementation of the core SMT algorithms. Since this project is more hands-on and demonstrative rather than theoretical, I'll be showing a bit of pseudocode but mostly writing actual algorithms in Python for the sake of the practical application of this translation technology on other minor languages.

For a more macro-level understanding of SMT's typical steps, **Figure 1** provides a brief overview of the modeling process. Normally, a language model would be applied after a set of possible translations has been generated by the algorithm, but that assumes a corpus of many more thousands of target language phrases and sentences are readily available. For the languages on which I'm focusing, they're simply not.

Here are a few basic definitions for the mathematical probabilities of traditional SMT as demonstrated by Brown et al. 1993 [2]:

$$p(e,a|f) \quad = \quad \frac{\varepsilon}{(l_f+l)^{l_e}} \quad \times \quad \prod_{l_f}^{j=1} t(e_j|f_{a(j)})$$

This is the critical part of the IBM Model 1 Algorithm: as **Figure 2** demonstrates, there are multiple possible combinations for a particular sentence's alignments at first. The above formula is the probability of an alignment for a source (French) sentence $a \mid f$ given an English sentence e, a normalization constant ε, considering the length of the English and French sentences l_e and l_f. This probability is over every word in our training sentence, so the probability is based on the product of the possibility of a given translation, the second product of the right side of the equation, where the function *t()* is the probability of a single word translation through which we iterate for each sentence.

This leave the question of how to efficiently (or not efficiently, if you prefer) calculate alignment probabilities over multiple training iterations. Once again, the great engineers at IBM Watson introduce EM, or Expectation Maximization. The algorithm for EM is as follows:

$$\prod_{l_e}^{j=1} \frac{t(e_j|f_{a(j)})}{\sum_{i=0}^{l_f} t(e_j|f_i)}$$

These algorithms, though they may look daunting, are remarkably simple. However, they require processing power and efficiency which is unavailable to many researchers to implement fully. My code takes a unique but simplified form of this algorithm that is easily-replicated and quite functional on any corpus size (though obviously, words that do not appear in the corpus will not be in the trained translation model). I took into consideration performance characteristics of the algorithm and determined that the most viable way of implementing it, which also guarantees acceptable quality. For the sake of efficiency and

Table 1: EM Application Statistics

n	approx probability sets	total time to iterate EM 5x
2	100,000	3 hours
3	300,000	11 hours
4	1,200,000	18+ hours

speed of modeling, both of which mean lower costs and are of chief concern to my research, I first tested a bigram model, which assumes two-word pairs as its maximum phrase length when training. This has advantages for speed but disadvantages for how much sense the overall translation makes, which the BLEU [1] scale cannot entirely measure.

I tested my algorithm on n-gram phrases where $n \in \{2,3,4\}$, ie. bigram, trigram, and quadgram phrases. Surprisingly, bigram translations seemed to work just as well as trigram translations when scored visually. In the interest of optimization, the above table shows the approximate time each model procedure took to complete.

Probability calculations depend on knowledge of the whole dataset and the sum of probabilities for all words and sentences: probabilities should always be in the range [0,1] and need to be neatly stored in memory (`float` types are inefficient for this, so a logarithmic approach, Perplexity, is generally preferred). This makes it more difficult to do multi-threaded processing of the training data. For this reason, my simplified implementation of the algorithm sums rather than multiplies probabilities for all fractional counts of n-gram alignment phrases, which still grants a similar degree of precision with much less overhead. The result, in n-gram form:

$$p(e,a|f) \quad = \quad \alpha \quad \times \quad \sum_{j=1}^{n} q(s_j|t_{a(j)})$$

In this formula, n is the n-gram word length, j is the alignment vector from source phrase s to target phrase t. Alpha α, like the normalization constants applied in previous models, is a set of fine-tuning metrics based on linguistic concerns that I discuss in the

subsections below.

Optimization Techniques: Fertility and Probability Reduction

I applied some additional metrics in order to better the data; the results that occurred most often often appeared as high-probability when they should not have. For example, inputting ”President” gave results ”das,” ”die,” and ”und” before ”Präsident.” This was fixed by algorithmically iterating through and finding the places where the words had the highest probability likelihoods, which is at ”the” and ”and” respectively, and deleting all other single-word instances of their translations.

In effect, this is like an application of IBM’s model 3 fertility model, and I have also applied that to my data to improve the results. Iterating through the dictionaries stored in `sourceWord: (targetPossibility:probFloat, targetPossibility:probFloat)` and lowering the score of those that have non-matching word counts can help for individual words like, President which ends up being translated as Mr. President, for example.

These optimizations are just a few that can be applied to improve the data, and I will continue to do more in order to better my results, but this is quite a compelling study in the efficiency of small-corpora and minimum-resources translation.

Results, Conclusion, and Discussion

While within minutes I was able to prepare a satisfactory set of 1,000 German and English UN sentences, Anglo-Saxon proved to be more divvicult. I could locate transcriptions of Genesis A and B and Exodus, all early Old-Testament passages, as well as supposedly-corresponding English translations, but could not locate texts with a sentence-for-sentence match, which is required for SMT to generate wordpair probabilities. Finding accurate digital version of texts which are rarely-researched and rarely-requested in any community,

especially for endangered or ancient languages, is a challenge well beyond the scope of this project.

The discussions here and my improved algorithm will be worthwhile for future applications of my own research and those who are looking to reliably build translation models on small text sizes and with minimal processing necessity on hand, meaning that this can be done with lower costs than the old, pre-IBM-Model days of rule-based translating using the archaic dictionary and manually-programmed grammar rules. The fact that one can even get translations on a 1000 word corpus is interesting, and proves that endangered languages still have hope.

Figure 1: Understanding Typical SMT Workflow

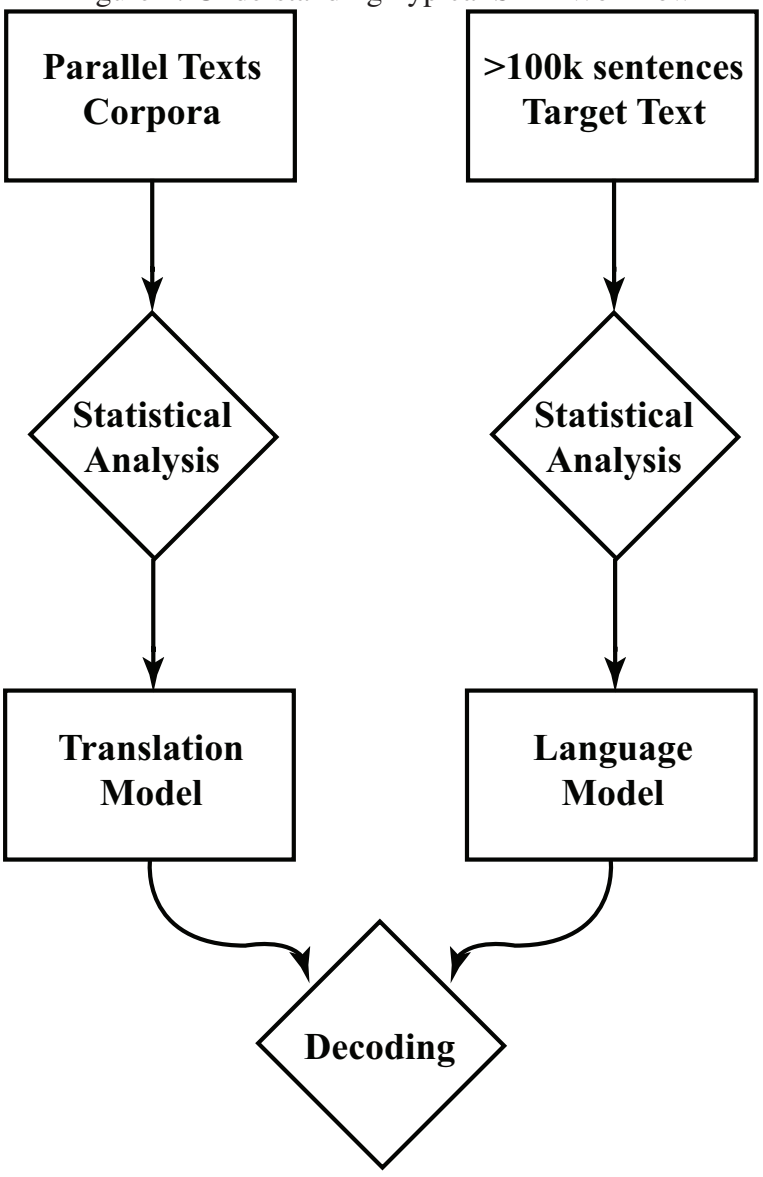

Figure 2: Expectation Maximization (EM) Algorithm Alignment Example

References

[1] IBM: Kishore Papineni et al, *Bleu: a method for automatic evaluation of machine translation*, (2001).

[2] Stephen A. Della Pietra Robert L. Mercer Peter F. Brown, Vincent J. Della Pietra, *The mathematics of statistical machine translation: parameter estimation*, Computational Linguistics **19**, 263–311.