Have you ever wondered how Google Translate works?

How do you teach a computer the many complex and dynamic rules of human language? Google's answer: you don't—there are simply too many grammatical rules, exceptions to those rules, and exceptions to those exceptions. Instead, let the computer learn the rules for itself.

By poring over millions of already translated documents, such as books, UN transcripts, etc., Google Translate detects patterns between the translation and original text that are unlikely to occur by random chance, and it uses those patterns to translate similar text in the future. This process is called **statistical machine translation**. Modern programs that use SMT consistently outperform programs that were taught how to translate by a human, i.e. rule-based translators, simply because a given set of rules cannot account for ambiguity in word meaning, idioms, and other idiosyncrasies.

My translator was also made using this same principle, albeit using a simpler algorithm than Google's. Essentially, given a **sentence *f*** in the source language (in my translator, German is the source,) we're trying to find a **translation *e*** in the target language (in my translator, English) that maximizes the probability p(*e*|*f*).

Those of you who have taken stats will appreciate this manipulation by Bayes' Theorem:

$$p(e|f) = \frac{p(e)p(f\|e)}{p(f)}$$

And since p(f) is constant, we get the following expression for the best English translation *eˆ:*

$$\hat{e} = \arg\max_{e} \; p(e)p(f\|e)$$

As you can see, two major components of the model are p(e) and p(f|e), known as the **language model** and the **translation model**, respectively. The third component, into which p(e) and p(f|e) are inputted, is known as the **decoder.**

The **language model,** p(e), determines whether the possible English translation *e* is good English. It does this by dividing the sentence into three-word units known as trigrams, and the probabilities of these trigrams, trained on data from an English text corpus, are multiplied together to get the overall probability p(e) for the entire sentence. For example:

p(I like bungee jumping off high bridges) = p(I |START START) * p(like | I START) * p(bungee | I like) * p(jumping | like bungee) * p(off | bungee jumping) * p(high | jumping off) * p(bridges | off high) * p(END | high bridges) * p(END | bridges END)

My predictive text function is a relatively simple extension of the language model's trigram (and bigram, and unigram/word) probabilities.

The **translation model,** p(f|e), determines whether the two strings are good translations of each other, trained on data from a parallel bilingual corpus between German and English. The specific algorithm I coded is called IBM Model 2, and without going into much detail, this model uses translation probabilities (e.g. the probability that "wir" translates to "we") combined with alignment probabilities (e.g. the probability that the 2nd English word aligns to the 3rd French word) to determine the overall probability of this sentence. Also, it uses an iterative process called estimation-maximization which is really cool; Wikipedia it if you have time.

IBM Model 2 is relatively outdated, but it is useful for quickly finding the best alignment between a German and English sentence, known as the Viterbi alignment, which I coded.

So now that we can score the likelihood of a translation, it begs the question: where are we getting said translations from? This is where the **decoder** comes in. I implemented a crude version of a decoding alogrithm called greedy hill-climbing, which essentially builds an initial seed translation, and then tries various word translation changes, swaps, joins, insertions, etc., until the score of the translation can't go any higher.

Instructions:

Click the buttons on the left panel to toggle between Translation, Predictive Text, and Sentence Alignment. Do not close the window. Be patient with the predictive text feature; it is trained over millions of words, so it takes a few seconds to update the predictions. It's also probably the coolest feature.

Resources

Pseudocode for IBM Model 1
http://www.ims.unistuttgart.de/institut/mitarbeiter/fraser/readinggroup/model1.htm
l
IBM Model 2- Coursera videos- https://www.youtube.com/watch?v=kj6izC-nQ00
Europarl corpus of English-German parallel text (download)
American National Corpus of English text (download)
Online paper for greedy hill-climbing decoder--
http://www.aclweb.org/anthology/N03-1010

I DON'T MEAN TO GO ALL LANGUAGE NERD ON YOU, BUT I JUST LEGIT ADVERBED "LEGIT," VERBED "ADVERB," AND ADJECTIVED "LANGUAGE NERD."