Machine Translation
A HMM alignment model based on IBM Model 1

Huizhan Lu

February 19, 2015

1. General idea

This assignment contains two parts, the IBM Model 1 and Hidden Markov Model(HMM).

First of all, the IBM Model 1 is an algorithm that generates the translation probabilities between an English word e and a French word f, i.e. $t(f|e)$. It assumes that the probabilities of translating from one English sentence to one French sentence only depends on $t(f|e)$, and thus is independent of the ordering of the two sentences.

Then, the Hidden Markov Model(HMM) is applied to find the relation in the ordering of adjacent words. We assume that the position a_j, of a French word f_j in target English sentence, depends only on the position of the previous English word a_{j-1}. In this assignment, we treat the output data of IBM Model 1 as the training data of HMM model, and use MLE estimate:

$$P(f_1^J, a_1^J|e_1^I) = P(J|e_1^I)\prod_{j=1}^{J}[a(a_j|a_{j-1}, I)t(f_j|e_{a_j})]$$

In the equation, $P(f_1^J, a_1^J|e_1^I)$ is the total probability from one English sentence to one French sentence. The $P(J|e_1^I)$ is the probability of generating a French sentence of length J, which is not needed. $a(a_j|a_{j-1}, I)$ is the transition probability from position a_{j-1} to position a_j, with the length of English sentence. Let $c(i - i')$ be the count of jumps of distance $i - i'$ in the training data,

$$a(a_j|a_{j-1}, I) = \frac{c(a_j - a_{j-1})}{\sum_{i=1}^{I} c(i - a_{j-1})}$$

and

$$t(f_j|e_{a_j}) = \frac{count(f_j, e_{a_j})}{count(e_{a_j})}$$

2. Implementation

When it comes to Implementation, the program can be split into several modules like follows:

i. Pre-training on IBM Model 1. Use an EM model to train on the dataset and get approximated results of $t(f_j|e_{a_j})$. There is one minor difference between my model and the Pseudocode in the lecture slides(Machine Translation IBM Model 1). In the Pseudocode we calculate $t(e|f)$ while in my program I used $t(f|e)$.

ii. Initialization of HMM Model. Initialize c from the results we get in the previous part.

iii. EM training on HMM Model. In Maximization part, I used Viterbi algorithm to ensure the program to converge faster. I also used negative log probability to ensure that the float number does not underflow. In Estimation part, use the new counts gathered from Maximization part, and calculate new parameters. The EM training on large dataset runs very slow and so I added a upper limit for iteration times.

iv. Predicting. Simply use the same equation to predict the outcomes.

3. Results

The AER for IBM Model 1 is 0.42, and for HMM model is 0.36.
(I used a data size of n = 10000 and Max iteration times = 20 for HMM model. An increase in either value may improve its performance)