Machine Translation HW2

Gary Qian (gqian1)

October 3, 2017

Code: https://github.com/GaryQian/MachineTranslation
Commit: 54fd1dffc88838e17fab5f0c5601ef61211bcfb3

1 IBM Model 1

I implemented an IBM Model 1 aligner, which was fairly straightforwards. It used a standard EM algorithm. I initialized the starting probabilities of each parings of words with a uniform distribution over all possible pairings, and ran the algorithm for 30 iterations.

IBM Model 1 attempts to optimize the parameters as defined by:

$$\theta = argmax_\theta \Pi_{n=1}^{N} p_\theta(f^n, a^n|e^n)$$

Where theta are the probability parameters, N is the number of sentences, and p_θ is the probability a French word f occurs given an alignment set a and English sentence e. We attempt to approximate this using an EM algorithm because the problem cannot be solved closed form. Thus, for each iteration, we will keep track of the instances each (f,e) word pairing occurs, as well as the total number of times the english word is paired with any french word. This will provide us with an updated p(f—e) by dividing: $count(f|e)/count(e)$. More formally, during the E step, we will find:

$$p(a_i = j|f, e) = \frac{p(f_i|e_i)}{\sum_{a_i=1}^{J} p(f_i|e_{a_i})}$$

We then update in the M step:

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

To output the final results, I simply found the highest probability pairing for each french word.

In my implementation, I decided to use a uniform distribution as initialization over all of the possible word pairings.

I found that my IBM Model 1 implementation produces an AER of 0.37-0.36 depending on how many iterations I ran. I experimented between 5 and 150 iterations and found that the AER lower bound for my implementation to be around 0.35.

2 Two-way independent (E to F and F to E) Model

I implemented a two way model that made use of two independently trained IBM Model 1 instances. The model had a double EM algorithm with separate probability distributions for each direction.

Let the $p_1(f|e)$ and $p_2(e|f)$ represent the two trained models using IBM Model 1. The final probability of any pairing was simply the product of these:

$$p_{final} = p_1(f|e)p_2(e|f)$$

This improved the AER performance significantly, dropping the AER down to 0.33-0.32 with iterations ranging from 5 to 150. The combined model outperformed a single IBM Model 1 significantly. However, the run time of this model became much longer.

3 Diagonal Weighing

I then applied a simple diagonal weighing function to the distributions to highly favor alignments close to the diagonal. Let λ = a constant found through experimentation. My new prediction function became:

$$p_{final} = p_1(f|e)p_2(e|f)(\frac{1}{|i-j|/\lambda})$$

I found though experimentation that a constant $\lambda = 8$ outperformed dynamically sized lambdas. The results of this were slight but noticeable, bringing the double IBM Model 1 from above's performance AER to about 0.32-0.315

4 Dice Coefficient Initialization

Previously, all of the probability distributions were initialized uniformly, however, I believed that there could be a fast and much better solution to determine the intitalization of the probability distribution. I attempted to apply the Dice Coefficient as starting points for the probability distribution. The idea is that the results from Dice are not completely accurate, so by starting with Dice's

Coefficient, we can get to a semi-trained state that would have otherwise taken much longer using a uniformly initialized EM algorithm.

After initialization, we continue to follow the EM algorithm, which is now starting at a state that is of AER 0.6 instead of a uniform distribution.

5 Two way IBM Model 2

To improve upon the results of the previous improvements, I implemented IBM Model 2, which now takes into account the length of the sentence as well as the probability of the word pairs in combination with the lengths of the sentences. This should improve upon the results of IBM Model 1. I also integrated my diagonalization improvements since that is only applied at the end and was relatively simple to add.

In Model 2, we follow a similar EM algorithm as Model 1 with a few key changes. We define $q(j|i, l_k, m_k)$ to be the probability the word pairing of i and j occur with sentences of length l and m, with k being the index of the sentence pair in the corpus. The probabilites found with IBM Model 1 are now represented as $t(f|e)$ to remain consistent with the work of Michael Collins which formed the basis of my Model 2 implementation. In the E step, we find a $\delta(k, i, j)$ "count weight" defined as:

$$\delta(k, i, j) = \frac{q(j|i, l_k, m_k)t(f_i^k, e_j^k)}{\sum_{j=0}^{l_k} q(j|i, l_k, m_k)t(f_i^k, e_j^k}$$

We then increment the counts by δ for each of the following counters:
$c(e, f),\ c(e),\ c(j|i, l, m),\ c(i, l, m)$

In the M step, we define the new q and t as follows:

$$t' = \frac{c(e, f)}{c(e)}$$

$$q' = \frac{c(j|i, l, m)}{c(i, l, m)}$$

My model 2 (single instance) resulted in a AER improvement to 0.301

Building upon the results of my experimentation above, I also implemented a double IBM Model 2 system that trains 2 models, one f-e and the other e-f and combined them in a similar fashion as above. The results were improvements on the models alone and brought the AER even lower.