Statistical Word Alignment

Joshan Bajaj and William Bernardoni

September 27, 2018

Contents

1 Improvements Implemented

1.1 IBM Model One

Our group started with implementing Model One as described on Professor Koehns slides. We chose to write our implementation in C++ for increased performance both time and memory wise, as well as for increased precision since we had more control over the data primitives (ie, we could 80 bit floating point precision numbers instead of 64 bit).

You can find our implementation of IBM Model One in the IBM_Model_1 folder. Instructions on running it can be found in that folders README.

1.2 MultiThreading and Convergence Measure

After implementing IBM Model One, our next step was to further improve our codes run time by implementing multithreading into our already existing algorithm. We wanted to be able to run our current model and future models to convergence at a threshold of 1, and so we felt speeding up the training of the language model was a necessity.

We measured convergence using the Cauchy measure of convergence, which is measuring convergence via the change between each point in a sequence rather than the distance of points in a sequence from the limit of the sequence – as we do not know the value the model is converging to, Cauchy seemed a better measure of convergence, as it is equivalent to the classical measure of convergence..

You can find our multithreading in any of our languageModel.cpp in IBM_Model_1 or the two i_IBM_Graph folders.

1.3 Naive Diagonal

The next improvement that we implemented into our model was to incorporate naive diagonal positioning as a metric in our alignment phase. This improvement is an idea that is used with IBM Model 2. Essentially, we wrote our model to weigh words that were at a diagonal from it more strongly as they were more likely to be the correct choices for the algorithm to pick as the right translation. However, this implementation of diagonal weighing is naive in that it doesnt affect the language model and that it doesnt take into account fertility. These two issues were addressed in our Graph Matrix Model and our Improved Graph Matrix Model.

The adjustment we used for our diagonal weighting was as follows:

$$2^{-|\frac{e}{|E|}-\frac{f}{|F|}|}$$

Where e and f are the respective indexes of the English and French words in the matching sentences.

Inspiration for this model came from this paper (http://aclweb.org/anthology/N/N13/N13-1073.pdf)

You can find our implementation of Naive Diagonal in the IBM_Naive_Positional folder. Instructions on running it can be found in that folders README.

1.4 Two Way Language

After implementing naive diagonal, our next improvement came from combining a French to English model with an English to French one trained on the same data. The idea to combine the two language models came from this paper (http://aclweb.org/anthology/N/N06/N06-1014.pdf), and while initially we found that certain weighing schemes reduced effectiveness, we eventually found one that provided useful results.

In order to combine the two models, we needed to get them to point in the same direction, and so once we had calculated $p(e|f)$, we used Bayes Theorem to get it pointing in the $p(f|e)$ direction as well. Our equation for Bayes was the general one, ie $p(e|f) = p(e) * \frac{p(f|e)}{p(f)} \rightarrow p(f|e) = \frac{p(e|f)*p(f)}{p(e)}$, keeping in mind $p(e)$ is 1.

Once Bayes Theorem was used to turn our $p(e|f)$ into another $p(f|e)$ value, we combined them by taking the square root of their product. We found that using this custom weighing scheme was more effective than taking the average because we wanted to use their findings when they agreed, but when they disagreed we wanted to be more harsh and more heavily weigh the lower models prediction.

You can find our implementation of Two Way Language in the IBM_F2E_E2F_Combine folder. Instructions on running it can be found in that folders README

1.5 Exact Match

In our previous models, we found that punctuation would sometimes map to a common word such as le. To get around this problem and to deal with others such as parenthesis, numbers, and proper nouns, we added a simple fix to our post processing alignment step. We added a check to see if a French and English word in the same sentence had identical string representations, and if they did, we ignored the language model and set their probability to 1 and then applied the distance effect as described in the Naive Positional section. We found that adding this change greatly improved our AER.

You can find our implementation of Exact Matching in the IBM_F2E_E2F_Combine_Exact, i_IBM_Graph, and i_IBM_Graph_2_Way folders. Instructions on running these can be found in their respective folders READMEs.

1.6 Graph Matrix

While we were implementing two way alignment, in parallel we were looking at ways to improve our general model. One of the key areas that IBM Model One failed at was in fertility. Before the Graph Matrix update to our model, we selected our alignments like so:

$$a(f, e) = \underset{f \in F}{argmax} p(f|e)$$

Where $e \in E$, the English sentence, and $f \in F$, the parallel French sentence.

This method however has some notable drawbacks. Some English words have multiple French words that should align to it firstly, and secondly it does not take into account what other words the French word may translate to. A French word could translate to multiple English words, but the less adjacent they are the less likely they are both from the same French word. Similarly, an English word could be from two French words, but the further apart they are the less likely they translate as a pair to the English word.

Seeing this, we wanted a model that allowed us to translate multiple English words to a French word and visa versa, and allow the choosing of those words to affect the probabilities of future word pair choices. We then formatted the question of choosing the best alignment for a sentence as a graph search problem, and created the below greedy algorithm.

```
Given English sentence E and French sentence F.
Construct a |E| x |F| matrix M, where each m[i][j] = p(f[j] | e[i])
While (true)
{
  for all m[i][j]
    if m[i][j] == -1
      For all other m[i][j] not equal to -1 in its row or
      column, multiply their probability by 1/2^d([i][j])
      where d([i][j]) is the distance to the nearest
      point adjacent to [i][j]
  Choose the largest m[i][j], which must be larger than some
```

```
  ε as an alignment and set it's probability to −1.

  If there are no probabilities larger than some ε, end the loop.
}
```

This greatly increased both precision and recall, and on inspection of the alignments had exactly the desired effect.

You can find our implementation of the Graph Matrix algorithm in the Graph_Align folder. Instructions on running these can be found in it's README.

1.7 Improved Graph Matrix

After improving our selection of alignments we decided to take a look at the model itself. We implemented two ideas from lecture.

First on each iteration of the EM algorithm, we added some smoothing to the adjustments of the probabilities - to simulate additional sentences having other potential alignments, and to make the EM algorithm get less excited about rare words (which it tended to overestimate the probabilities assigned to rare words and use them as sort of a garbage collector for anything it was unsure of.)

The second improvement we made, is we incorporated our distance function as described in the Naive Diagonal section into the model itself. On each of the count steps of EM, we multiplied the count by the same distance metric described earlier. This was to heavily encourage the model to converge on a mostly diagonal representation of the data, as the alignments between French and English seem to mostly be diagonal.

You can find our implementation of these improvements in the i_IBM_Graph folder. Instructions on running these can be found in it's README.

1.8 Improved Model with Two Way Language Model

After completing all of these improvements, we decided to merge our two branches of models, and try running the Two Way Language model with our Graph Matrix algorithm and the improved model. The results we got were a resounding improvement over all of our prior models.

2 Results

These were our results with each model trained on the full dataset to a Cauchy convergence of 1.0, tested on the given dev set.

Model	Precision	Recall	AER
IBM_Model_1	0.657	0.778	0.300
Naive Positional	0.666	0.837	0.276
Graph Matrix	0.747	0.864	0.212
Improved Graph Matrix	0.765	0.873	0.197
Two Way Language	0.847	0.763	0.191
Two Way with Exact	0.847	0.766	0.190
Improved Graph Matrix with Two Way	0.908	0.804	0.139