Experimenting with IBM Model 2 Style Word Aligners

David Philipson and Nikil Viswanathan

{pdavid2, nikil}@stanford.edu
CS224N Fall 2011

Introduction

In this project, we build an alignment model which suggests probabilities for alignments between two sentences. For set up, we have two sentences a source sentence and a target sentence. We assume that the two sentences should have the same meaning in two different languages, and we are interested in translating from the source sentence language into the target sentence language. We define an alignment as a way of assigning each word in the source sentence to a word in the target sentence. Our model will be able to return the probability of a given alignment, or produce a single most likely alignment when given a sentence pair. This model, alongside a language model, can be used to translate sentences from the source language to the target language.

Approach

We implemented three models. The first model is based on pointwise mutual information and can only produce a predicted optimal alignment, but not return the probability of an alignment. This makes it unsuitable for working as part of a larger translation model, but we could still examine it's results as a kind of baseline. For this model, we simply count the number of times each word in one language appears in a sentence pair with each word of the other language (as well as the number of times each word appeared individually), and normalize these counts to produce empirical probabilities. Then to produce an "optimal alignment" given a sentence pair, we send each source word *f* to the target word *e* which maximizes P(*e*, *f*) / (P(*e*) P(*f*)).

The second model is the IBM "model 1." That is, we make the simplifying assumption that for a given source word *f*, a priori the index to which that source word aligns in the target sentence is uniform (and not, for example, dependent on what *f* is). We must then learn the parameters p(*f* | *e*) for each source word *f* and target word *e*, which we do via the EM-algorithm.

Finally, we implement the IBM "model 2." This makes the slight complication to model 1 that the a priori probabilities for where a source word *f* aligns are no longer uniform, but differ based on the position of *f* relative to the candidate positions in the target sentence (but are still independent of what *f* is).

Correctness

For the pointwise mutual information model, there is nothing to verify for correctness, as it makes no claims regarding probability.

For the IBM models, we must check that the alignment probabilities, returned by the getAlighnmentProb function, properly sum up to 1 over all alignments on a given sentence pair. Our model 1 computes this probability as follows:

- For each source word *f* in the source sentence, we take the stored probability P(*f* | *e*) for the target word *e* to which it is aligned.
- We divide this by the sum of P(*f* | *e*) over every target word *e* in the sentence.
- This gives a probability for each source word *f*. We multiply all of these together (actually by adding in log-space) to get the overall alignment probability, which we return.

Now consider the sum of these values over all alignments between the sentence pair. By factoring in a manner similar to that discussed in Knight's paper,[1] we see that this quantity can be expressed as a product over source words *f* of the sums over target words *e* of the quantity computed above:

$$\prod_f \sum_e \frac{P(f \mid e)}{\sum_{e'} P(f \mid e')}$$

It is clear that each summation equals 1, and so the product does as well. We must also be sure that the individual probabilities are always non-negative. For this, we must be sure that the stored values of P(*f* | *e*) are all non-negative. But this is guaranteed by our algorithm, as we shall check in the next paragraph.

For the EM algorithm to work, we must have that the computed values P(*f* | *e*) are valid probability distributions at each step. That is, for each target word *e*, we must have the sum over all source words *f* of P(*f* | *e*) to equal 1. We ensure this by normalizing after each step of EM. That is, each time we collect the partial counts, for each *e* we divide all the counts c(*f* | *e*) by the sum of all such counts.

Computations for model 2 are analogous, but with an extra factor of the distortion probability wherever P(*f* | *e*) appears. All the above reasoning still holds.

Analysis

We spent significant effort optimizing our reverse distortion probability function. Initially we saw the distribution of buckets converge to all NULL_WORD favored alignments after running our EM algorithm. In our analysis below, we show how we tried various algorithms for modifying this distribution to look more like one we expected to see from the data. Initial attempts shifted the probability mass to the bucket on the opposite end and after trying out several different strategies we finally reached a distribution centered around zero distortion, which also predictably produced the best results.

Bucket Distribution

As we tested out different numbers of buckets, bucketing schemes, and ways to handle the null alignment, we saw several different distortion distributions of weights over the alignment

[1] Knight, Kevin. A Statistical MT Tutorial Workbook. MS., August 1999.

buckets. Our goal, based on our hypothesis of alignment distributions was to get a distribution would be fairly steep around zero and still have a smooth distribution in the buckets nearby. In almost all of the distributions we saw from our model, we noticed that there was a steep peak with a single bucket or at most a couple of buckets (usually always < 3) taking all of the weight. We tried several methods to smooth out the distribution and in particular tested out increasing the size and number of buckets so that different distortion alignments would fall in different segments. We discovered that the spiked distributions centered around zero distortion had good word alignment performance scores. Our best results in fact, came from a bucketing distribution which had the null alignment and the zero distortion alignment receiving all of weight.

Null Bucket

We tried several approaches for handling the null bucket. First we tried learning the null weight implicitly as part of the bucketing scheme. Second we tried assigning a separate bucket for null alignments and learned an individual weight for that. Finally we used a static weight for the null alignment as was suggested in the handout. We saw mixed results with all of these approaches, with an overall small variance (from ~.42 - ~.37 AER) and observed that the results also depended pretty substantially on our choice of the number of buckets and the bucketing scheme.

Learning Null Implicitly

Our first approach just treated the null bucket as a normal distortion possibility and bucketed it into whichever bucket the particular alignment hashed to in the bucketing function. Our rationale was that rather than handpicking the weight for null, we could just learn the weight from EM algorithm and this would produce a better estimate of the actual probability than a guess. Using the first bucketing function (described below) we achieved a AER of around 0.416 (depending upon the number of buckets that we used for these tests).

Learning Null Separately

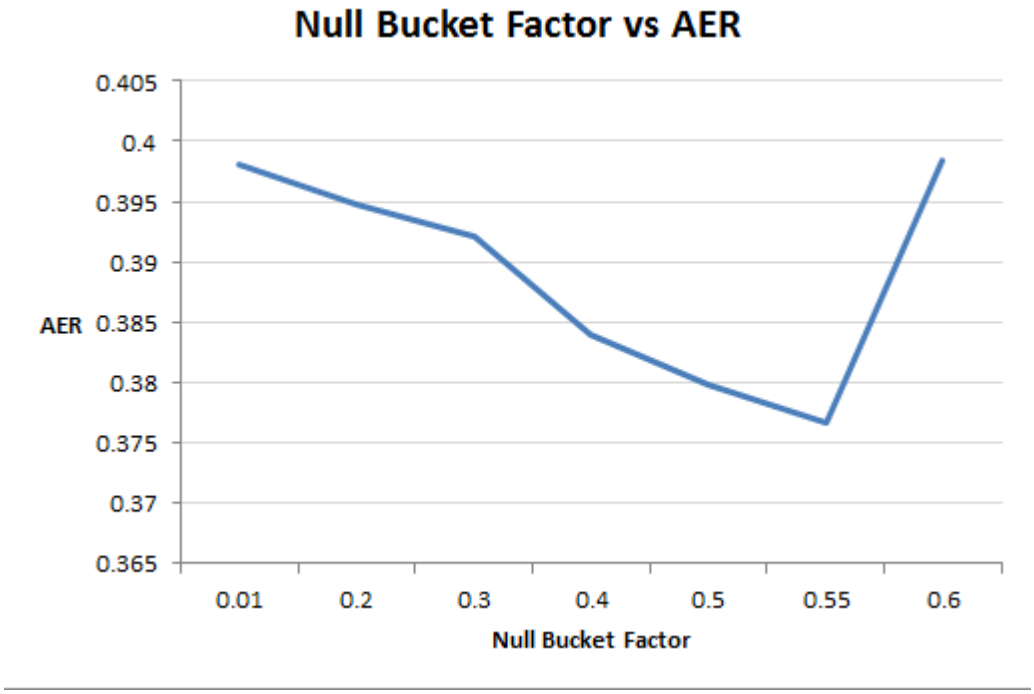
Our second parallel approach put the null alignment into its own bucket but still normalized it with the other buckets and attempted to learn the value from EM. This new approach yielded a slightly better AER (shown in the table below) and depending on the number of bucket and their size overall seemed like a better alternative. The intuition behind this approach was that we were not sure what the weight of the null alignment should be so we learn the weight, however we still want to distinguish the null alignment from other alignments which have the same distortion length as they are semantically different.

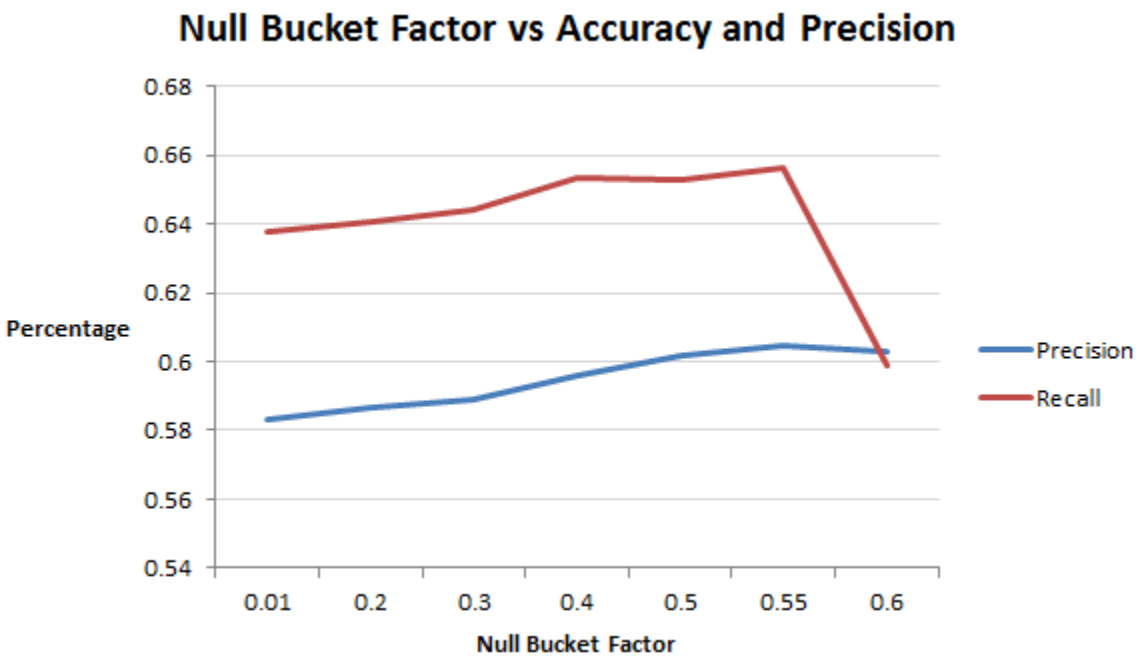
Buckets	7	9	13
Precision	.552	.584	.557
Recall	.631	.638	.587
AER	.421	.398	.419

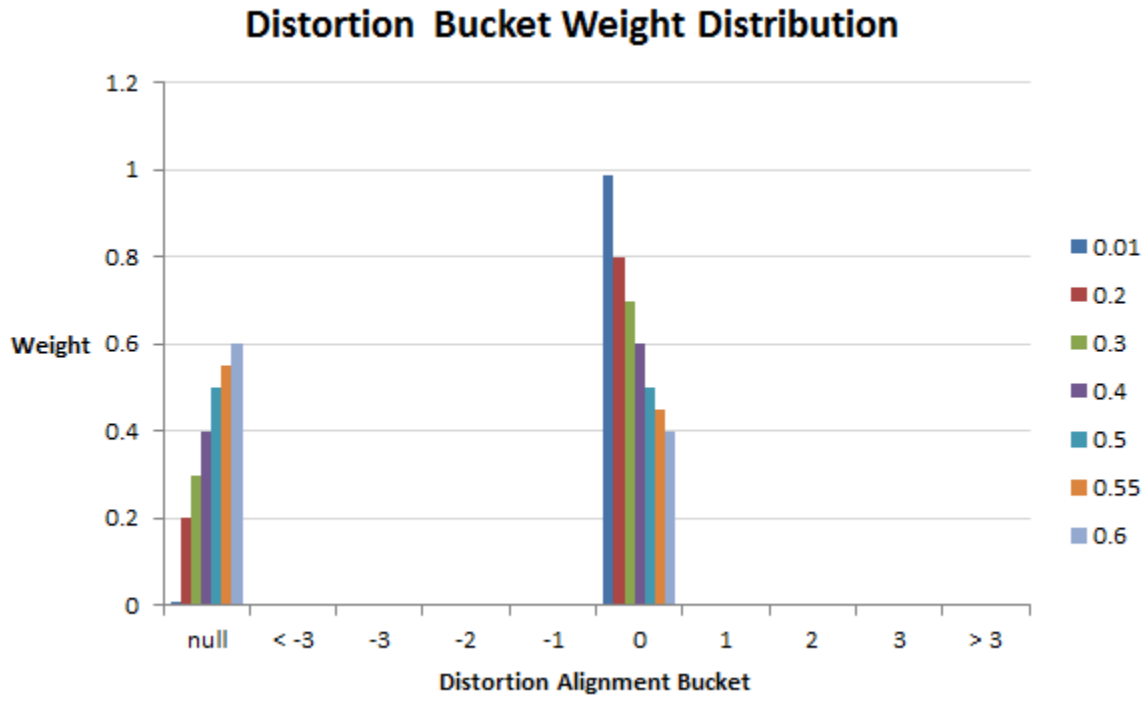
Table 1: Learning weights for the null bucket

Fixed Null

Finally we decided to try out the simplest method of just fixing a null factor. To our surprise, this method worked fairly well. The data showed that increasing the weight for null, far past the suggested values of .1 or .2, resulted in a increase in the (AER) performance. As seen in the table of data below, this effect continued up until assigning about 55% of the weight to the null bucket. Our hypothesis is that since the original bucketing distribution is so peaked around the zero distortion, assigning a larger value to the null bucket allows some of the variation that could be captured by a more smoothed out distribution.







Bucket Size

Varying Number of Buckets

In addition to varying the size of the buckets and our approach for handling the null alignment,

we tested out different combinations of numbers of buckets in conjunction with the previous factors.
The data is shown below, and we found that slightly varying the number of buckets had a reasonable effect on the overall performance (as measured by the AER score) with a swing of about 3 AER points and that having 9 buckets actually worked the best for our first bucketing function as described below.

Buckets	7	9	13
Precision	0.553	0.584	0.577
Recall	0.631	0.639	0.586
AER	0.421	0.397	0.42

Bucketing Functions

In combination with the other features that we tested, we also tried out a couple different bucketing functions. The primary forms we experimented with were:

$$1.\, bucket = roundToInteger(englishPosition - frenchPosition\,(englishLength\,/\,frenchLer$$

(with a separate bucket on each end to catch any distortions larger than the maximum amount)

$$2.\, bucket = MAX[\backslash_?]DISPLACEMENT\ *(englishPosition\ /\ englishLength - frenchPositio$$

Both functions had different settings which gave varying results in terms of performance and they exchanged performance wins when tweaked these settings. We decided to go with the second one as with multiple different sets of parameters we saw slightly improved results with respect to the former function even over its best parameter set we could generate.

Results

Overall, we experimented with several different parameter sets in terms of the number of distortion alignment buckets, the bucketing function, the size of the distortion buckets and handling null values. The most significant gains came from implementing the improved ibm models and a smaller amount came from the parameter tuning and learning methods we tried. We improved the model a good 5-8 % in terms of AER from the baseline model two with our initial formation for the distortion weights. Our best alignment AER: .377

We found that our model gave very high weight to the central distortion bucket. When examining its outputs, we found that it had extremely high accuracy on alignments which were either on the main diagonal or within one of the main diagonal. On the other hand, it frequently missed alignments which were far from the diagonal. This suggests that our model's emphasis on the center distortion bucket allowed it to specialize for certain types of sentences.

As examples of this phenomenon, observe the following two predicted alignments.

In each case, the model successfully selects almost all of the gold standard alignments which are on or near the diagonal. However, in the second case we can see that the model made a number of incorrect alignments in the lower right, where the correct alignments were actually far from the diagonal. We therefore believe that a more advanced model, which would use features to recognize cases where off-diagonal elements are more likely, would be more successful than our implemented model.

Data from decoding

With model one model, we get:

WER: 0.8191858630482802
BLEU: 0.013819843319345021

With model two model, we get:

WER: 0.7154654854652464
BLEU: 0.026463548634564654