CS224n PA1

Justin Cunningham
jcnnghm@stanford.edu

Abstract

In programming assignment 1, I implemented a pointwise mutual information (PMI) word aligner, as well as IBM Model 1 and IBM Model 2 word aligners. The PMI Aligner achieved an AER of 0.6837, while Model 1 and Model 2 achieved 0.4314 and 0.3775 respectively. In the second part of the assignment, I implemented a feature for Phrasal which was able to achieve a 15.761 average BLEU score.

1 Part 1

In Part 1, I implemented PMI and IBM Model 1 and IBM Model 2. AER results for all three models, for French-English, Hindi-English, and Chinese-English are available in Table 1 and Table 2. Model 1 and Model 2 consistently outperformed PMI. Model 2 outperformed Model 1 for French-English and Chinese-English, but underperformed Model 1 for Hindi-English on both the dev and test sets.

All of my implementations in Part 1 subclass a `BaseAligner` class that I introduced. `BaseAligner` provides an implementation of the `align` method and provides a helper that, when given a `SentencePair`, returns a new list containing the target words and a null word, ensuring that the placement of the null word is consistent. The `align` implementation calls the `calculateAlignmentProb` method which `BaseAligner` defines, with the expectation that subclasses will implement it. The `align` method aligns source-target word pairs with the highest calculated probability, using `argMax` from `Counter`. `BaseAligner` essentially acts as a framework, allowing the other aligners to implement only the code that is unique to that particular alignment model.

1.1 PMI Aligner

The PMI Aligner collects source word counts, target word counts, and combined source and target counts in the `train` method. Before completing training, the model collects a total count for source and target words, which is helpful to reduce the `align` runtime, since collecting these counts repeatedly can be expensive.

To calculate alignment probabilities, the following function is used:

$$a_i = \arg\max_j \frac{p(f_i, e_j)}{p(f_i)p(e_j)} = \arg\max_j \frac{p(e_j|f_i)p(f_i)}{p(f_i)p(e_j)}$$

The probabilities are trivially obtained from the counts previously collected.

1.1.1 Error Analysis

The PMI Aligner clearly isn't very good. The aligner tends to both align too many words to a word, and align words to completely wrong words. It's not uncommon for nearly all of the source words to align to a single target word. I believe one of the issues is tie breaking. If multiple alignments are equally probable, the `argMax` function will always choose the same one.

In addition, rare target words, like "affirmative", "essentially" or "growers" tend to have lots of alignments, presumably because the low probability in the denominator is causing a higher overall alignment probability.

Finally, if a target word appears in the sentence multiple times, all of the source words that align to it will align to the same target word, ignoring word order differences.

1.2 IBM Model 1 Aligner

The `IBMModel1Aligner` code is very simple, with most of the complexity pushed into the `Model1ExpectationMaximizer` helper, which deals with the EM algorithm and initial

	French-English	Hindi-English	Chinese-English
PMI	0.6837	0.8376	0.8444
Model 1	0.4314	0.5838	0.6534
Model 2	0.3775	0.5901	0.6238

Table 1: AER results for development set.

	French-English	Hindi-English	Chinese-English
PMI	0.6215	0.8309	0.8519
Model 1	0.3994	0.5473	0.6559
Model 2	0.3265	0.5651	0.6435

Table 2: AER results for test set.

Iteration	AER
1	0.6989
2	0.4772
3	0.4487
4	0.4382
5	0.4362
6	0.4381
7	0.4314
8	0.4305
9	0.4314
10	0.4314

Table 3: Model 1 AER

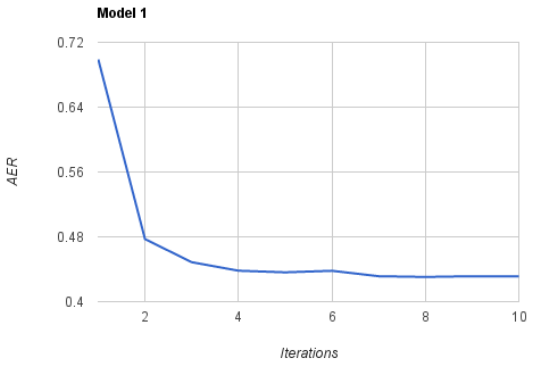

Figure 1: Model 1 - AER and Iterations

setup. This design makes it easier for the IBM Model 2 code to initialize with the output of IBM Model 1, and to share the same general EM algorithm.

The `Model1ExpectationMaximizer` is initialized with uniform $t(f_i|e_j)$ probabilities. During each iteration of my algorithm, I update the count $C(f_i|e_j)$ by adding $\delta(k,i,j)$, where $\delta(k,i,j) = \frac{t(f_i^k|e_j^k)}{\sum_{j=0}^{l_k} t(f_i^k|e_j^k)}$ for every source-target word pair in each sentence pair k (algorithm is Collins Figure 4). To improve the runtime efficiency, I calculate the divisor first, outside of the loop. At the end of each iteration, I normalize the counts.

I run my algorithm for 10 iterations, because as can be seen in Table 3 and Figure 1 the rate of improvement of the AER is fairly flat at that point.

The distortion probabilities in Model 1 are uniform, so I used $a_i = \arg\max_{j \in 0..l} t(f_i|e_j)$ to calculate alignments after the model is trained.

1.2.1 Error Analysis

Model 1 produces much more reasonable alignments than PMI. One of the most obvious weaknesses that remains is alignments with repeated source words. Again, because of the way `argMax` resolves ties, all repeated source words will align with the same target word. This can make it appear that a sentence with three commas in French, will only have a single comma in English. In the absence of repeated words (or interchangeable words), the model seems to cope well with word order differences. It appears more common to align too many words to a word, rather than completely wrong alignments.

There is some garbage collection appearing, with the abbreviation "Mr." aligning to the period. Both "M." and "Monsieur" appear in the same sentence, so it's possible that the probability of aligning "Mr." with french translations is relatively spread out, while the period appears in most every sentence that "Mr." does.

1.3 IBM Model 2 Aligner

The `IBMModel2Aligner` code is very similar to the IBM Model 1 code, with the main differences being an updated alignment probability function to include distortions, as well as running IBM Model 1 first and using the resulting t values to initialize Model 2.

Again, most of the complexity is pushed into the `Model2ExpectationMaximizer` helper. The helper re-uses the Model 1 EM code, updating the probabilities and counts to include distortions. I implemented a method `getQKey` to manage building keys for the q probabilities, to make it simpler to maintain a consistent index order.

The `Model2ExpectationMaximizer` is initialized with $t(f_i|e_j)$ from running the `Model1ExpectationMaximizer`. $q(j|i,l,m)$ is initialized uniformly. During each iteration of my algorithm, I update the counts $C(f_i|e_j)$ and $C(f|i,l,m)$ by adding $\delta(k,i,j)$ to each, where $\delta(k,i,j) = \frac{q(j|i,l_k,m_k)t(f_i^k|e_j^k)}{\sum_{j=0}^{l_k} q(j|i,l_k,m_k)t(f_i^k|e_j^k)}$ for every source-target word pair in each sentence pair k (algorithm is Collins Figure 2). As in Model 1, I calculate the divisor first, outside of the loop. At the end of each iteration, I normalize both counts.

I run my algorithm for 10 iterations, because as can be seen in Table 4 and Figure 2 the rate of improvement of the AER is fairly flat at that point.

To calculate alignments after the model is trained, I used $a_i = \arg\max_{j \in 0..l}(q(j|i,l,m) \times t(f_i|e_j))$.

1.3.1 Error Analysis

Model 2 offers subtle improvements over Model 1, but nothing nearly as dramatic as the move from PMI to Model 1. Model 2 is definitely still experiencing some garbage collection, where, for example, many function words align to the English word ruling, but none with the French word decision. It definitely still appears that there's a tendency for too many words to align with a single word.

The addition of distortion probabilities appears to have helped with some of the multiple alignments, and it appears that most of the duplicate words now align to a word relatively close to them. It would be interesting to weight the distortion and translation probabilities, to see what kind of impact could be had by tuning their relative importance. Examining the Hindi alignments seems to suggest that increasing the importance of the distortion probabilities could result in better alignments, since many of the missed alignments are pretty far from where they'd be expected to fall on the diagonal.

Iteration	AER
1	0.4607
2	0.3960
3	0.3837
4	0.3775
5	0.3851
6	0.3869
7	0.3841
8	0.3803
9	0.3803
10	0.3775

Table 4: Model 2 AER

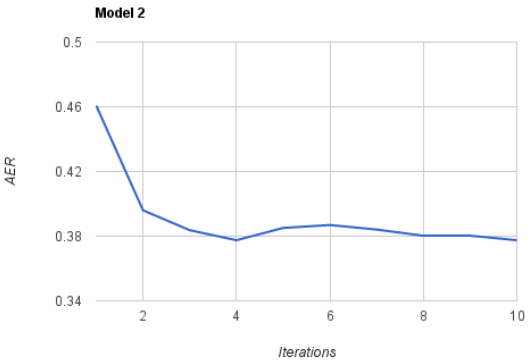

Figure 2: Model 2 - AER and Iterations

Between Model 1 and Model 2, the precision increased from 0.5183 to 0.5758, and the recall from 0.6746 to 0.7219. This relatively modest increase seems to mirror what I have observed intuitively, the alignments are marginally better, but when they're incorrect, the issues are very similar to Model 1.

2 Part 2 - Phrasal Feature

The feature that I implemented added indicators using parts of speech data extracted from the source and target phrases using CoreNLP. The total number of weights learned increased from 10 in the baseline to 6,065. My measured Phrasal baseline was 15.4059, and I measured an increase to 15.7609, a difference of 0.355, over 50 runs.

My belief was that some of the target phrases

weren't really grammatical, so I thought that by comparing parts of speech in the source and target, and learning weights for parts of speech in the target, the system would be able to generalize a little more about English sentence structure.

2.1 Features

2.1.1 Bias

I added a simple indicator bias feature to each rule regardless of source or target phrase, to act as a constant. The system learned a weight of -0.0572 for this feature. I expected the bias to be negative, to counteract the weights that I believed would be learned from parts-of-speech co-occurrences.

2.2 Target Parts of Speech

I added unigram, bigram, and trigram part of speech indicator features over the target phrase. That expectation was that the system could learn in general which English parts of speech work well together.

For example:

```
TARGET_POS_TRIGRAM:IN:NNS:IN
```

2.2.1 Part of Speech Co-Occurrence

I created an indicator feature pairing every part of speech in the source with every part of speech in the target. The part of speech tags for English and French aren't consistent, so this kind of approximates matching nouns to nouns, for example.

For example:

```
POS_CO_OCCUR:JJ:ADJ
```

2.2.2 Part of Speech Change

This indicator feature outputs sorted part of speech tag sets for both the source and target phrases. I decided to sort the tags and put them in sets instead of displaying the tags in the order they appeared to avoid over-fitting.

For example:

```
POS_CHANGE:DT|IN|VBZ:CLS|DET|ET|P|V
```

2.3 Output Impact

A few examples are highlighted in Table 5 and Table 6 - in general, where there is improvement, the sentence tends to follow more common English-language patterns.

With Feature
the gdp of germany has a anticipated growth 0.5 % , that of france 0.4 % .
they have won 1:0 in montenegro and celebrating the qualification for 200 million
in the czech republic , only 23 % of families of my students eat at the same time .
they are not really mistakes , but the expression of an incomprehensible laziness on the part of the developers .

Table 5: With Feature Translations

Baseline
the germany 's gdp growth expected 0.5 % , that of france of 0.4 % .
they have won in montenegro 1:0 and celebrating qualification for 200 million
in the czech republic , only 23 % of families of pupils my eat at the same time .
they are not really mistakes , but the expression of an laziness incomprehensible on the part of the developers .

Table 6: Baseline Translations

2.4 Error Analysis

In evaluating, sometimes it appeared that parts of speech matching had won-out over "better" phrases that were previously selected, probably because they aren't very common. On the other hand, it appeared that more natural phrases were selected overall, probably preferring grammatically correct uncommon phrases.

I believe the biggest room for improvement would be adding a featurizer that ran over the full or partial target sequence, creating parts of speech trigrams there as well. Examining the likelihood of transitions between phrases seems like it would be pretty fertile ground.

References

Manning, Christopher D. and Surdeanu, Mihai and Bauer, John and Finkel, Jenny and Bethard, Steven J. and McClosky, David. 2014. The Stanford CoreNLP Natural Language Processing Toolkit. *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pp. 55–60.