CS224N PA1: Machine Translation

Andrew Giel
BS Stanford 2015, Computer Science
agiel@stanford.edu

Jonathan NeCamp
BS Stanford 2015, Computer Science
jnecamp@stanford.edu

I. Introduction

This paper presents the results of three separate models for producing word alignments from parallel texts in an unsupervised manner. Presented are the PMI Model, IBM Model 1, and IBM Model 2. Additionally, we present a new feature for the Phrasal system that increases BLEU score.

	French-English	Hindi-English	Chinese-English
PMI	.7708	.8975	.8748
Model 1	.3674	.5916	.5852
Model 2	.3193	.6186	.5878

AER for the development set (10k)

	French-English	Hindi-English	Chinese-English
PMI	.7489	.8419	.8466
Model 1	.3556	.5874	.5952
Model 2	.3094	.5992	.5780

AER for the test set

II. PMI Model

Our Pointwise Mutual Information (PMI) Model is a step in the right direction as compared to the Baseline Model as it actually utilizes the bitext to find alignments, but it still a very poor alignment model. The pros of this model are that it trains quickly, as it only has to pass over the data once in order to make predictions. Unfortunately, this speed comes at the cost of effectiveness. This model makes a very naive and straightforward assumption about alignments by choosing the alignment that maximizes the joint probability of e and f normalized by the product of the probabilities of e and f separately. This model seems especially inadequate when dealing with uncommon words. A typical problem with PMI is epitomized in *Fig. 1*. Nearly every English word was predicted to align with the French word "ressentir". The IBM Models provide a significant improvement on this rudimentary model.

III. IBM Model 1

Our implementation of the famous IBM Model 1 is fast and effective, employing multiple techniques to ensure both proper estimation of probabilities and quick convergence.

While researching how to implement this model, we compared approaches employed by the two readings presented by the course staff: Michael Collins' paper "Statistical Machine Translation: IBM Models 1 and 2" and Kevin Knight's tutorial "A Statistical MT Tutorial Workbook". Although effectively doing the same task (estimating the hidden variables of alignments via Expectation-Maximization), the manner in which these two readings present the algorithm differ slightly. In the end, we chose to follow the Collins paper, as we felt the implementation approach presented was faster than that of the Knight tutorial. Note that in each Expectation (E) step of Knight one must: 1) compute $P(a, f|e)$ by multiplying the t probabilities, 2) normalize $P(a, f|e)$, resulting in $P(a|e, f)$, and then 3) weight ts by $P(a|e, f)$. On the other hand, the Collins approach requires each E step to update each $c(e_i, f_j)$ by $\delta(k, i, j)$. Also note that with this approach we do not need to compute $c(e_i)$ since we can leverage the Counter class to normalize the t values in the M step. From an implementation point of view this is a trade off between memory usage and speed; the Collins method requires us to store extra information in the form of the c values along with the t values, yet is much faster as it requires less computation per step when compared to the Knight method. By implementing this model in the manner presented by Collins we acknowledge this trade-off of increased memory usage for increased speed, but we stand by this decision as our model iterates (and subsequently converges) very quickly. Reflecting our choice of implementation, the notation in this paper matches that used in the Collins paper.

One of the most crucial design decisions made was determining when to stop the iterations of the Expectation-Maximization (EM) algorithm. In order to ensure good estimation of the translation probabilities, we chose to establish that the algorithm had converged when the average change in probabilities was less than some small value ϵ or the number of iterations surpassed some value T. More formally,

$$converged = 1\{\Delta_{avg}^{(t)} < \epsilon \vee t > T\}$$

where

$$\Delta_{avg}^{(t)} = \frac{\sum_i \sum_j |t(f_j|e_i)_t - t(f_j|e_i)_{t-1}|}{\sum_i \sum_j 1}$$

where t is the current iteration. This definition of convergence ensures that the $t(e|f)$ values, or translation probabilities, are at a stable point in their estimation via EM. We ran multiple experiments in order to find a suitable value for ϵ by varying the value and observing the change in AER for the French, Hindi, and Chinese development sets (see Figure 2). While running this convergence experiment we noticed that for very low values of ϵ that the AER actually increased, meaning our model performed worse than at the previous value of ϵ. The

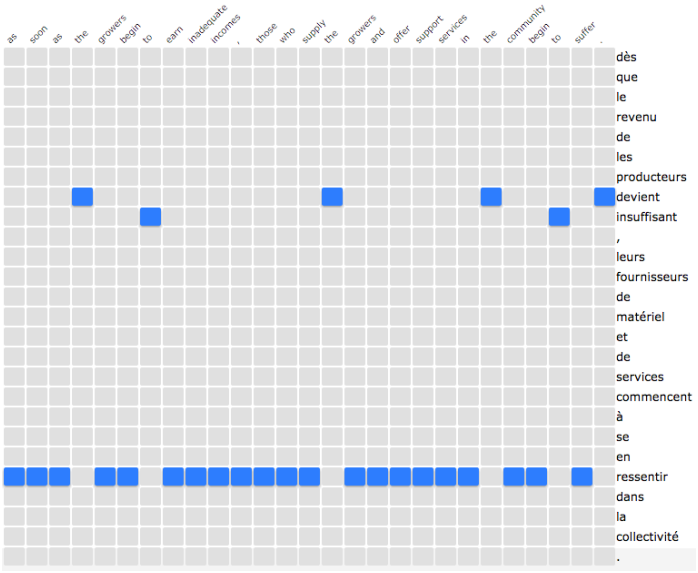

Fig. 1. PMI Predicted Alignment

only logical conclusion to draw from this result is that at very small values of ϵ we actually overfit to our training data. As a result, we decided to choose a moderate value of ϵ to allow for proper convergence without overfitting. We concluded the optimal ϵ to be 0.001. Notice in Figure 2 the relatively large drop in AER between 5×10^-2 and 10^-3 across all three languages. We also found that even at very low ϵ values, t rarely surpassed 30, and as such set T=30.

One key implementation feature incorporated in order to

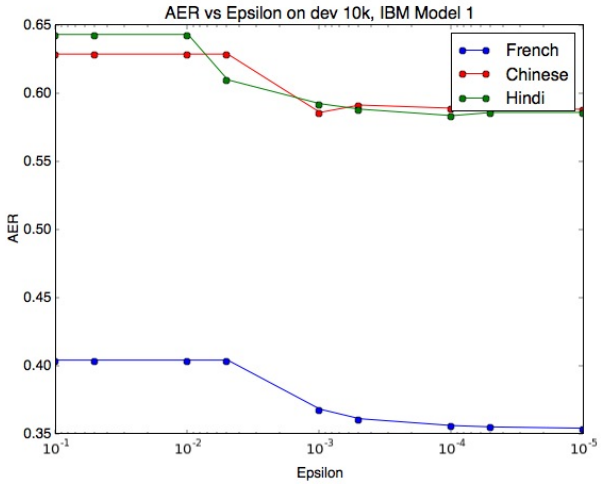

Fig. 2. AER versus different values of ϵ for IBM Model 1

throttle memory usage in the instantiation step was to faux-initialize all the $t(f|e)$ values. In the first E iteration, each $t(e_i, f_j)$ should be initialized to some uniform value u. In this E step for each value of $c(e_i, f_j)$ we update by $\delta(k, i, j)$ which is a function of the t values, which in this iteration are all uniform. Note

$$\delta(k,i,j) = \frac{t(e_i|f_j)}{\sum_{j=0}^{l^{(k)}} t(e_i|f_j)} = \frac{u}{\sum_{j=0}^{l^{(k)}} u} = \frac{1}{l^{(k)}}$$

As such, we did not even have to initialize the t values for the first iteration, as long as we assumed they were uniform the δ values would follow this form. This little implementation trick helped us to speed up the first iteration and ensure sparsity in our probabilities, leveraging the Counter class given to us and throttling our memory usage.

On top of this optimization, we noticed an opportunity to decrease the computational complexity of calculating $\delta(k, i, j)$ by calculating the denominator only once for each target word. Note that the denominator of $\delta(k, i, j)$ is $\sum_{j=0}^{l} t(e_i|f_j)$ and as such is the same for each target word (e_i). Therefore, when incrementing the $c(e, f)$ values by δ, we examine each target word first, calculate the denominator of δ for that target word, and then calculate the numerator which is a function of both the source and target word. This approach makes the calculation of all δ values for a given pair of parallel sentences to be lm (where l is the length of the source sentence and m is the length of the target sentence) as opposed to l^2m. Although l and m should be modest in size, we feel this optimization played a significant role in the speed of our implementation.

IBM Model 1 is a significant improvement on the PMI Model. IBM Model 1 is essentially the same algorithm as IBM Model 2. Thus, the problems with Model 1 occur in Model 2. We defer error analysis to the next section where the deficiencies with both IBM Models will be highlighted.

IV. IBM Model 2

IBM Model 2 is extremely similar to IBM Model 1. The most salient difference between the two models is that Model 2 iteratively updates the $q(a_i|i, n, m)$ parameters based on expected alignments rather than assuming uniform q's, as is the case in Model 1. Additionally, Model 2 initializes its t parameters using the parameters learned in Model 1. As is mentioned in Collins, Model 1 converges to a global optimum, and the resulting parameters have been shown in practice to be great initializers for Model 2.

Other than these minor differences, the algorithm for Model 2 is essentially the same as Model 1. The t parameters are initialized using Model 1, but the q parameters are initialized randomly. For each Expectation step, expected alignment counts, $c(e_i, f_j)$ and $c(j, i, n, m)$, are collected. Since Model 2 does not assume uniform q parameters, the expected alignment probability, δ, is calculated as follows:

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

The $c(e_i, f_j)$'s are then normalized in the Maximization step to yield the t parameters and and $c(j, i, n, m)$'s are normalized to give the q parameters.

Because of the similarity between the two IBM models, the optimizations mentioned in the Model 1 section also apply to

our implementation of Model 2. However, one area of slight difference was convergence criteria. Since Model 2 does not assume a uniform probability for all q parameters, one must check that both q and t parameters have adequately converged. Thus, we compute the Δ_{avg} for both q and t and check that they are both below a determined ϵ before exiting the EM iteration. More formally,

$$converged = 1\{(\Delta_{avg}^{(t)} < \epsilon \wedge \Delta_{avg}^{(q)} < \epsilon) \vee t > T\}$$

As with Model 1, we ran Model 2 over the development set multiple times to determine ϵ. For Model 2, we determined $\epsilon = .005$ to be the best choice because as shown in *Fig. 3*, AER values remained stagnant for values after .005. Furthermore, we preferred this method of determining convergence as opposed to a fixed number of iterations because we noticed that different languages converged at varying number of iterations. More specifically, Hindi tended to require more iterations for $\Delta_{avg}^{(t)}$ and $\Delta_{avg}^{(q)}$ to fall below ϵ.

As we assumed it would, our IBM Model 2 was the best performing model of the three that we implemented. IBM Model 2 was a significant improvement on the PMI Model and a slight improvement on IBM Model 1. Model 2 works well but is certainly not perfect. Like the PMI Model, it has difficulty with alignments when it encounters an uncommon word. *Fig. 4* is a good example of this. It accurately predicts alignments for most words but inaccurately predicts alignments to the less-seen word "mobilis". Additionally, the model does a poor job of finding spurious words. Of the 37 sentences evaluated in development, it predicted 0 alignments to NULL word. The heuristic we used, insert the NULL word into every source sentence and then treat just like any other word, certainly has deficiencies and is an area for future improvement. For example the heuristic does not account for the fact that alignment to the NULL word should be equally likely from any index in the target sentence.

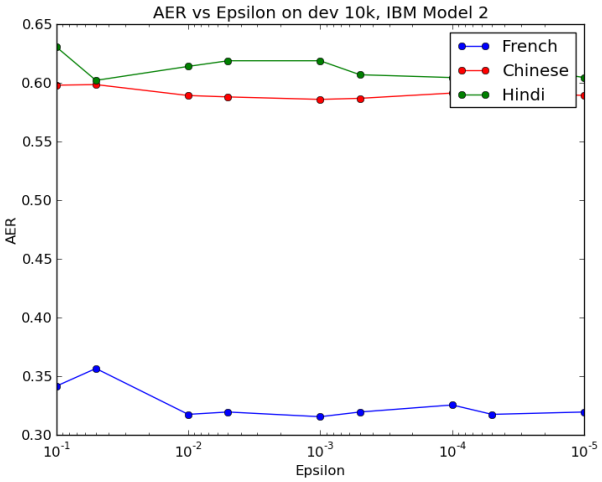

Fig. 3. AER versus different values of ϵ for IBM Model 2

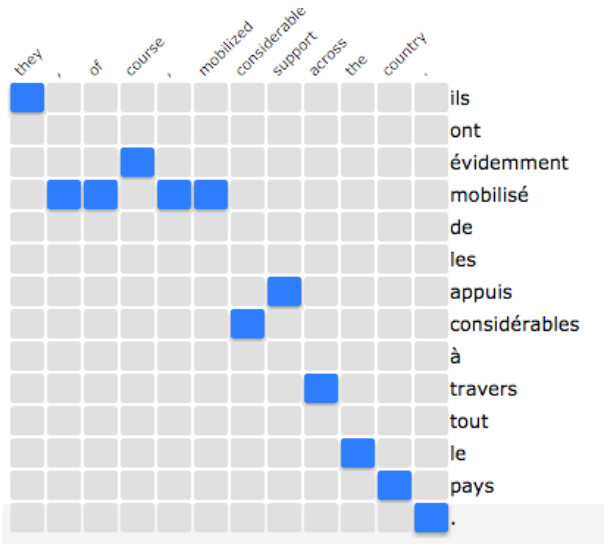

Fig. 4. IBM Model 2 Predicted Alignment

V. Phrasal Feature

In order to supplement the Phrasal baseline system, we experimented with a number of different features to be incorporated into the log-linear model. These features include Levenshtein Edit Distance, Difference in syllables (based on a simple heuristic), Differences in number of letters between source and target phrases (which we will refer to as Letter Difference), Absolute Letter Difference, Difference in number of words between source and target phrases (which we will refer to as Word Difference), and Absolute Word Difference. Each of these is grounded in some intuition we have regarding the nature of the task.

Utilizing Levenshtein Edit Distance was based on the notion that English and French often have cognates or translations with similar word bases. The idea was that Levenshtein Edit Distance would give the model this knowledge; low edit distances correlate to similar looking phrases between the two languages. Upon experimenting with this added feature, we encountered mixed results. In some trials this feature gave us some of the highest BLEU improvements of any feature(+.6 from baseline), while in others it severely hurt our BLEU score (-.3). Overall, the feature seemed to give about a +.3 increase in BLEU but we concluded that this feature was too unstable to be used reliably.

We used a simple heuristic to estimate the number of syllables in both source and target phrases and used the difference of these values as a feature. The heuristic we used was to count all vowels in the phrase, counting adjacent vowels only once and not counting 'e' when at the end of a word. This feature's intuition is that if we can give the model information on the relative difference in the number of syllables spoken between French and English it may be able to use this information to make better translations. For example, assume that in general French words have more syllables than English words. With this information the model

may be able to reject bad translations in which the English phrase has more syllables. This is simply an example, but the idea remains in that we hoped the model could systematically assess the validity of a translation aided by syllable count differences. In reality, this feature was fairly helpful, but averaged only +.2 BLEU score above baseline. As such, we chose to explore other feature options.

Another feature we experimented with was the difference (both relative and absolute) of number of letters in the source and target phrases. This feature is similar to the syllable count feature, as we want the model to have the information about the word length differences in the two languages. When comparing this count between source and target phrases we saw an increase in BLEU of +.25 above baseline. We also experimented with taking the absolute value of this difference, but found that this made the model perform worse on average (+.21 increase in BLEU). This can be easily explained, as taking the absolute value of the difference actually gives less information on the relationship of character counts between the languages. For example, assume in training that French had many more characters than English. In a testing environment when taking the absolute value of this difference, an English phrase with 2 more letters than its proposed French translation looks the same to the model as an English phrase with two less letters than its French translation. This ambiguity explains the drop in BLEU between the two features. Despite the intuition and performance increase of this feature, in the end we chose to implement another feature: Word Difference.

Word Difference follows the same intuition as many of the other features presented here; the feature gives more information to the model about the difference in word counts between the two languages. This feature directly relates to the concept of alignments, as it encapsulates the notion of one-to-one, one-to-one, many-to-one, one-to-many, and NULL mappings by supplying the model data on these differences. On average, this feature resulted in an increase of +.29 BLEU above the baseline while also exhibiting low variance (unlike Levenshtein Distance). Once again, we experimented with using the absolute value of this count as a feature but also saw a drop in BLEU as Absolute Word Difference achieved an increase of +.22 BLEU above th baseline on average. This can be explained with the same logic applied to Absolute Letter Difference above. Due to the relatively high increase in BLEU score, stability of increase, and its accompanying logic, we chose Word Difference as our feature to supplement the Phrasal System.

We found it interesting to compare the translations of our system with the Word Difference feature to those of the baseline system. In general, it seems that our feature causes many more one-to-many mappings from French to English than the baseline system. Most of the words inserted as a result of this are functional words as opposed to content words. For example, here is a translation of the sentence 'il aurait fallu 226 voix pour l' approuver' to English via the baseline system: 'it would take 226 votes to approve it' and here is the translation via our system incorporating Word Difference: 'it would have taken 226 votes to approve it.' As one can see, our system added the term 'have' to the translation. This changes the tense of the entire sentence, implying the vote already happened and required 226 votes in the past as opposed to a more hypothetical sentence claiming the vote would in theory require 226 votes. This distinction is subtle but requires the system to have knowledge of different verb conjugations, which can result in the many-to-one mappings from Romance languages such as French to English. Our feature seemingly gives the model more flexibility to add in functional words to better express different conjugations, and as such seems to serve its intended purpose.

VI. Conclusion

In conclusion, we have implemented, tested, and analyzed three distinct yet related algorithms for translation and alignment. We also extended a baseline Phrasal system by adding a feature which we call Word Difference. The task of Statistical Machine Translation is not a trivial one, and as such there is undoubtedly room for improvement. Nonetheless, we hope that this paper is evidence for the potential in this field attainable by applying seminal algorithms, engineering techniques, and domain knowledge.

References

[1] Collins, Michael. 2011. Statistical Machine Translation: IBM Models 1 and 2.

[2] Knight, Kevin. 1999. A Statistical MT Tutorial Workbook.