CS 224n Programming Assignment 2: Machine Translation

Todd Sullivan
todd.sullivan@cs.stanford.edu

Pavani Vantimitta
pavani@stanford.edu

1 Introduction

For PA2 we implemented three word alignment models and tested them with the provided decoder to produce translations of French sentences into English. Our first model was a simple PMI-based, surface-statistics model. Our other two models were IBM Model One and Model Two. We experimented with several variations of the IBM models including various bucket sizes and mapping techniques for Model Two's distortion probability distributions and various parameters for the decoder.

We produced a learning curve for each model with training sets ranging from 447 sentence pairs to 220,824 sentence pairs. Model One and Model Two achieve their lowest AER on the 221k set with 0.280 and 0.174 respectively. They achieve their highest AER on the 447 set with 0.516 and 0.367 respectively. We achieve our best BLEU-4 score of 0.139 with Model Two using our best language model from PA1 and the default decoder settings. In the following sections we use f to denote the source language/words and e to denote the destination language/words.

2 Implementation Details

In this section we will briefly cover the implementation of our models and the tricks/tweaks used to speed up mass evaluations of models. We skip discussion of our implementation of the Baseline Replacement model, which is the PMI-based surface-statistics model, because it simply involves adding counts in a similar manner to the baseline word aligner.

For both Model One and Model Two, we calculate the probability of an alignment a for a sentence pair (f, e) as

$$P(f, a \mid e) = \prod_{i=0}^{len_f - 1} P(a_i \mid i, len_f, len_e) P(f_i \mid e_{a_i})$$

where the special NULL word in the target language is at position $a_i = -1$, $P(a_i \mid i, len_f, len_e)$ is the distortion probability, and $P(f_i \mid e_{a_i})$ is the translation probability. Both models use the same notion for the translation probability, and only differ by their distortion probability distributions, as described below.

2.1 Model One

In Model One we give the NULL word a fixed distortion probability of 0.2 and distribute the remaining 0.8 uniformly across the destination words in the sentence. Thus $P(a_i|i, len_f, len_e)$ equals 0.2 when a_i is -1 and $P(a_i|i, len_f, len_e)$ equals $0.8 / len_e$ otherwise. We tried several values for the fixed NULL distortion probability and found that the suggested value in the assignment handout (0.2) worked best.

Aside from varying the fixed NULL distortion probability, we also tried completely removing the distortion probability from the equation. This has a slightly different effect from a full uniform distribution (which would have all distortion probabilities, including the NULL word's, as $1 / (len_e + 1)$) because we use the distortion probability in our EM training. Thus without the distortion probabilities, the counts that are collected are not scaled by the length of the sentence.

Table 2.1 shows the AER scores and BLEU-4/WER decoding scores for using the fixed NULL + uniform method and completely removing the distortion probabilities. Decoding used the baseline unigram MLE language model with a pseudo-count of one for the NULL word. As the table shows, including a distortion probability as we described above gives slightly better performance over completely removing the distortion probability from the model.

	AER									BLEU-4	WER
Training Size	447	5k	12k	33k	52k	73k	94k	113k	220k	80k	80k
Fixed NULL	0.52	0.38	0.35	0.32	0.31	0.30	0.29	0.29	0.28	0.026	0.752
No Distortion	0.51	0.40	0.38	0.35	0.34	0.33	0.32	0.32	0.31	0.016	0.800

Table 2.1: AER scores and BLEU-4/WER decoding scores.
Decoding used the baseline unigram MLE language model with a pseudo-count of one for the NULL word.

2.1.1 Training

We train our Model One word aligner using the EM algorithm as described in lecture. We maintain a CounterMap t that holds the translation probabilities, which are probability distributions across the source language conditioning on the destination language. We use our modified version of the Counter class, which allows us to set a zero item mass attribute that is the value returned when the key does not exist in the counter. t's first key designates the destination word, while the second key (the key for each inner Counter object) designates the source word. Thus each Counter object within the Counter Map is a conditional probability distribution.

We start the training by inserting a Counter object into t for every word in the destination vocabulary (plus the NULL word) and setting each counter's zero item mass to be uniform ($1 / len_f$). Thus on the first run, all attempts to access the value within t given the two keys fails and the uniform probability is returned. After initializing t, we begin our EM loop.

First, we create a new CounterMap called tcounts that uses the same mapping of keys (destination word, then source word). We loop through all sentence pairs (f, e), adding values to tcounts. For each word f_i in the source sentence, we compute the total probability mass for the word being aligned with any word in the destination sentence:

$$total_i = \prod_{j=-1}^{len_e-1} P\left(j \mid i, len_f, len_e\right) P\left(f_i \mid e_j\right)$$

Then for each (e_j, f_i) pair in the sentence (including NULL as an e_{-1}), we add $P(j|i, len_f, len_e)P(f_i|e_j)/total_i$ to the pair's running value in tcounts.

After calculating all of the tcounts, we recalculate t by normalizing each counter in tcounts and writing the result to t. After normalization, we repeat the loop that begins with the previous paragraph. If we have just finished the first iteration through this loop, we also set the zero item mass of each counter within t back to 0 and prune our probabilities by remove all destination/source word pairs from t where the probability is less than 10^{-21}. This pruning saves memory and allows us to run additional tests quickly as described in Section 2.4. We repeat the EM loop until the largest change in absolute value for any value within t is less than 0.001.

2.1.2 Finding the Best Alignment

When finding the best alignment in the alignSentencePair function, we choose the alignment for each source word independently of the other source words. We align each source word f_i to a destination word according to the following equation:

$$a_i = \arg\max_{j \in \{-1,1,\ldots,len_e-1\}} P\left(j \mid i, len_f, len_e\right) P\left(f_i \mid e_j\right)$$

2.2 Model Two

Model Two extends upon Model One by including a distortion probability that is learned during EM. In Model One, the distortion probability was a fixed 0.2 for aligning to the NULL word and uniform across the remaining 0.8 for the words in the destination sentence. In Model Two, we again fix the distortion probability of aligning to the NULL word, but to 0.255. We discuss evaluating the fixed NULL distortion probability and attempting to let EM choose the value in Section 2.2.4.

Aside from training and maintaining the distortion probability distributions that are conditioned on the source word position, source sentence length, and destination sentence length (which we will describe below), Model Two is the same as Model One. The alignSentencePair method is the same as Model One except for the different definition of the distortion probabilities.

With the exception of the NULL word (which has a fixed distortion probability of 0.255), our distortion probability distributions are of the following form:

$$P\left(a_i \mid i, len_f, len_e\right) \propto d\left(bucket\left(a_i - i\frac{len_e}{len_f}\right)\right)$$

Here, bucket is a function that maps its input (real numbered displacements normalized for overall sentence length) to an index in the array d as suggested in the assignment handout. We studied the effect of varying the amount of buckets (amount of indices in d) and the effect of an absolute value mapping (e.g. -5 displacement and +5 displacement both mapping to the same bucket) or having separate buckets for negative versus positive displacements. We discuss mapping methods in Section 2.2.4 and bucket sizes in Section 2.2.5.

2.2.1 Training with EM

We train and calculate the distortion probabilities by maintaining a probability distribution over index values that the bucket function returns. Our array d holds this probability distribution. In the beginning of training, we initialize the t values to the values calculated by Model One's EM training performed with the same training set. We initialize the d array by setting it to a Gaussian-looking distribution centered on the zero displacement bucket. During the EM loop, we keep track of dcounts, which has one collector for each index in d, in a similar manner to our tcounts. dcounts is actually just an array of doubles that is the same size as the array d.

At the beginning of an EM iteration, we initialize our dcounts and tcounts to zero. We then loop through each sentence pair, gathering tcounts in the same manner as before. We collect dcounts in the same way as tcounts (both are adding the same value to some bucket or location), except that in dcounts we add the value to the collector for the index that the bucket function returns for the specific source/destination pair. Thus dcounts are accumulated by:

$$dcounts\left(bucket\left(a_i - i\frac{len_e}{len_f}\right)\right) \mathrel{+}= \frac{P\left(j \mid i, len_f, len_e\right)P\left(f_i \mid e_j\right)}{total_i}$$

After collecting the counts, we calculate the new t values from the collected tcounts in the same manner as in Model One. We calculate the new probability distribution d by normalizing the dcounts array (calculating the sum of the array and then dividing all values by that sum). Unlike Model One, we repeat the EM loop for 50 iterations instead of waiting for the largest absolute change to drop below 0.001. We tested waiting for the change to drop below 0.001 and found that the tiny improvement over quitting after 50 iterations was not worth the drastically longer processing time.

The next two sections discuss how our distortion probability distributions are actually calculated using the probability distribution over buckets that is held is d. Section 2.2.2 describes our initial, flawed attempt that did not result in proper probability distributions but still produced good results. Section 2.2.3 discusses our fix to create proper probability distributions and our mechanism for handling all of the distributions.

2.2.2 Incorrect Probabilities

In our initial attempt at Model Two we simply used the probability distribution stored in d as the distortion distribution for all sentence pairs. I.e., for all distortion probabilities except the NULL word we had

$$P\left(a_i \mid i, len_f, len_e\right) = (1 - 0.255)d\left(bucket\left(a_i - i\frac{len_e}{len_f}\right)\right)$$

Unfortunately, this does not result in proper probability distributions because for a given i, len_f, and len_e, summing over all possible destination word locations a_i may not hit all indices of d exactly once and the summation over all values in d equals one. Thus our first attempt did not result in proper probability distributions. Nevertheless, the values that this first attempt produced were closely correlated to the correct method in Section 2.2.3, resulting in much better AER scores than Model One.

2.2.3 Correct Probabilities and Efficiently Handling Them

The correct way to generate the distortion distributions is to calculate a separate distribution for each tuple (i, len_f, len_e). Thus for a fixed tuple (i, len_f, len_e), we generate a probability distribution that has 0.255 probability for $a_i = -1$ and the remaining 0.745 distributed over the values $a_i = 0, 1, \ldots, len_e - 1$ as follows:

$$total = \sum_{j=0}^{len_e - 1} d\left(bucket\left(j - i\frac{len_e}{len_f} \right) \right)$$

$$P(a_i \mid i, len_f, len_e) = 0.745 \cdot \frac{d\left(bucket\left(a_i - i\frac{len_e}{len_f} \right) \right)}{total}$$

We manage the probability distributions using a hash map from strings to arrays of doubles and a function getDistortionProbability that given the values (a_i, i, len_f, len_e) returns the correct probability. When getDistortionProbability is called, if a_i = -1 then we return 0.255. Otherwise, we check our hash map for the tuple (i, len_f, len_e). If the tuple exists then we retrieve the tuple's probability distribution as an array and return the value at the a_i index. If the tuple does not exist then we generate the distribution as described in the previous paragraph and store the distribution in the hash map.

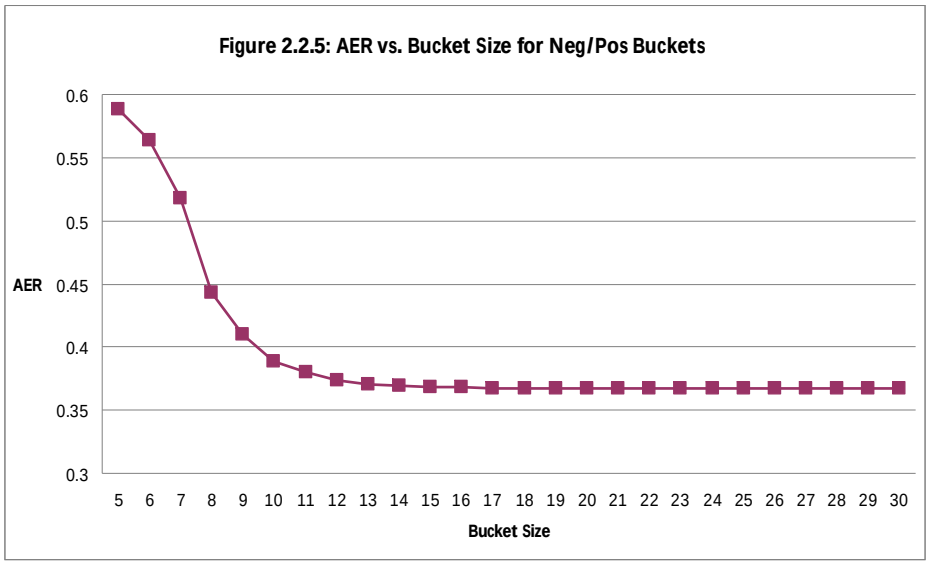
2.2.4 Absolute or Neg/Pos Buckets?

We tested using an absolute value mapping in the bucket function and having negative and positive values map to separate buckets. In the absolute buckets case, we mapped negative values to the same bucket as their respective positive value (e.g. a distortion of -5 was mapped to the same bucket as +5). In the neg/pos buckets case, we had separate buckets for forward and backward distortions. We chose to use the neg/pos bucket method due to the results of the next section and because we can imagine a pairing of languages negative distortions are quite frequent while positive distortions are not (or vice versa). In such a setting, the absolute buckets method would not capture this phenomenon.

2.2.5 How Many Buckets?

We examined the effect that bucket size (the amount of buckets) has on AER. We evaluated the AER score on the base set of 447 sentence pairs as we varied the bucket size from 5 to 30. In the absolute buckets case, a bucket size of 5 means that we have 6 actual buckets (0, 1, 2, 3, 4, 5). In the neg/pos buckets case, a bucket size of 5 means that we have 11 actual buckets (-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5). Our tests concluded that as the bucket size increases, both methods converge to the same AER of 0.355 but the neg/pos buckets converges faster by about four bucket sizes.

Figure 2.2.5 shows the AER of the neg/pos buckets method as a function of the bucket size. As seen in the graph, AER converges to 0.355 by a bucket size of 13. The poor performance of small bucket sizes was due to too many alignments being mapped to the buckets on either end. With a bucket size of 5, there were too many alignments that were less than -5 or greater than +5 and thus go mapped to the -5 or +5 bucket respectively. Through watching the probability distribution d during training, we found that this caused EM to place all of the probability mass on the zero distortion bucket and end buckets. We chose to use a bucket size of 20 as a safe buffer in case the AER converges less quickly when using larger datasets.



2.2.6 Null Distortion Probability

We chose the NULL distortion probability by selecting the smallest value that resulting in Model Two choosing a perfect alignment on the miniTest dataset. We tried letting EM choose the NULL distortion probability by adding an extra bucket to d that was indexed only when aligning to the NULL word. The result was that EM chose a probability of zero for the NULL word. This did not translate into good AER values, so we kept the fixed probability of 0.255.

2.3 Decoder Issues

Due to smoothing issues and our pruning method with t to save space, Model One's getAlignmentProb function sometimes calculates a probability of zero for a given alignment during decoding. Since smoothing was not a focus of this project, we solved the problem by returning the value closest to zero that a double can hold when the computed probability was zero. Since we used Java, this value returned was Double.MIN_NORMAL.

2.4 Other Details

In order to quickly evaluate our models under different conditions and settings (such as the LM, WA, and length weights in the decoder), we saved the result of training with each dataset to a file. Our models identify the training set by the amount of sentence pairs in the set and the language ordering of the set, which was generally "french-english" in our case. If the model had previously trained with the training set then the model loads the probability distributions from the proper file. If the model has not trained with the training set then the model trains as we previously described and outputs the result to the appropriate file. Due to our pruning method, a Model One distribution file for a 51k training set only requires 11 MB of space (without compression) and for a 220k pair set only requires 50.5 MB of space (without compression. A Model Two file only requires 6.9 MB (uncompressed) and 22.1 MB (uncompressed) respectively because there are more close-to-zero probabilities to that our pruning method removes.

3 Proper Probability Distributions

In this section, we skip the superficial statistics model and show that our Model One and Model Two probability distributions sum to one.

3.1 Model One

$$\sum_{a_i=-1}^{len_e-1} P(a_i \mid i, len_f, len_e)$$

$$= P(-1 \mid i, len_f, len_e) + \sum_{a_i=0}^{len_e-1} P(a_i \mid i, len_f, len_e)$$

$$= 0.2 + \sum_{a_i=0}^{len_e-1} \frac{0.8}{len_e} = 0.2 + 0.8 \sum_{a_i=0}^{len_e-1} \frac{1}{len_e}$$

$$= 0.2 + 0.8 \sum_{a_i=1}^{len_e} \frac{1}{len_e} = 0.2 + 0.8 = 1$$

$\sum_{f \in Vocab(F)} P(f \mid e) = 1$ because each Counter object within the CounterMap tcounts is renormalized after completion of EM to produce the conditional probability distributions and our CounterMap uses the destination word as the first key. So each Counter object actually represents our desired conditional probability distribution after normalization.

For a given destination sentence e of length I, the Model One generative store assumes that we first pick a length J for the source sentence, then choose an alignment between the destination and source sentences, and then for each source position j we choose a word f_j by translating the destination word that is aligned to it. This works out to give a probability of a source sentence being aligned in a particular manner to the given destination sentence as $P(f, a \mid e) = \prod_{i=0}^{len_f-1} P(a_i \mid i, len_f, len_e) P(f_i \mid e_{a_i})$, which by summing over 'f' and 'a' sums to one because the inner probability distributions are proper probability distributions.

3.2 Model Two

$\sum_{a_i=-1}^{len_e-1} P(a_i \mid i, len_f, len_e) = 1$ since our computation of the conditional probability distribution in the getDistortionProbability function involves pulling values from the array d and then normalizing the values.

$\sum_{f \in Vocab(F)} P(f \mid e) = 1$ because each Counter object within the CounterMap tcounts is renormalized after completion of EM to produce the conditional probability distributions and our CounterMap uses the destination word as the first key. So each Counter object actually represents our desired conditional probability distribution after normalization.

Thus by the same reason as Model One, the total probability distribution sums to one because the inner probability distributions (the translation and distortion distributions) sum to one.

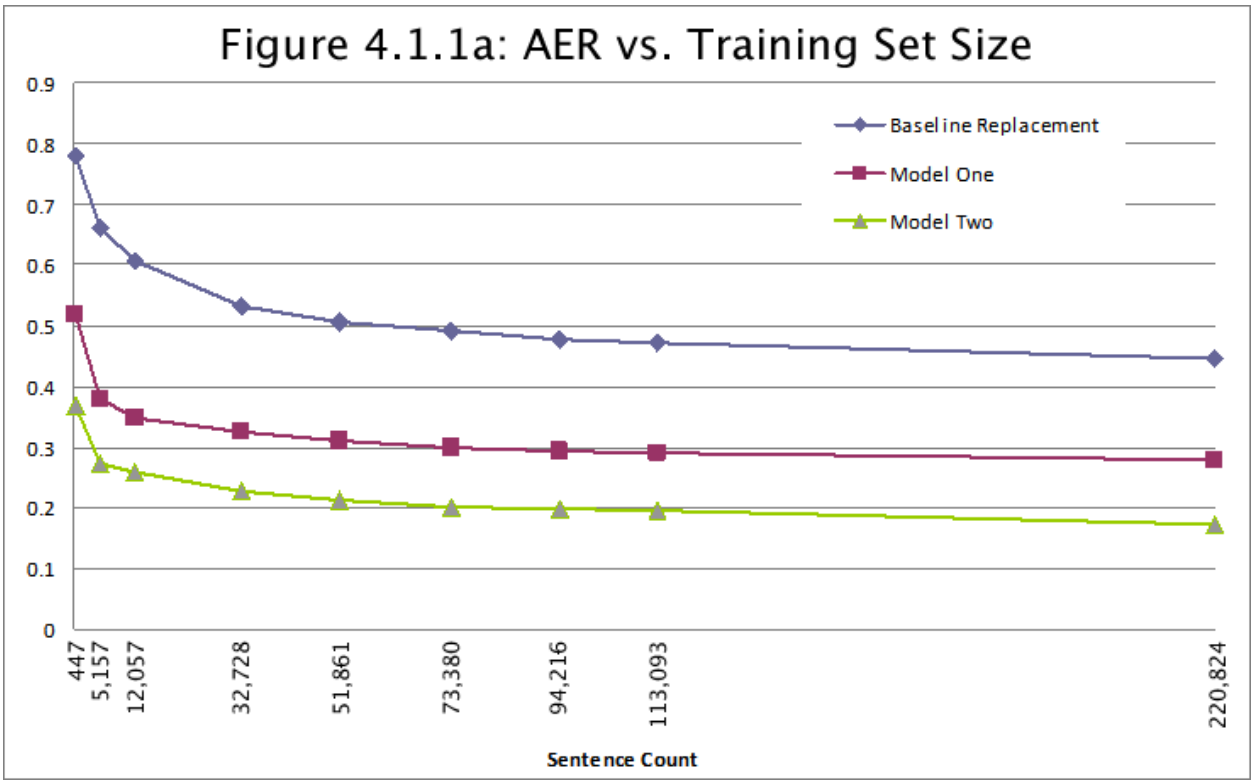
4 Performance Analysis

4.1 Word Alignment

In this section we present learning curves for each of our models and alignment examples that showcase the strengths and weaknesses of Model One and Model Two. While we

4.1.1 Learning Curves

We ran Baseline Replacement (the PMI-style model), Model One, and Model Two on a series of increasing training set sizes to generate learning curves for AER, precision, and recall on the test set. Our data set sizes range from 447 sentences to 220,824 sentences. Figure 4.1.1a shows the AER of each model as a function of training set size.



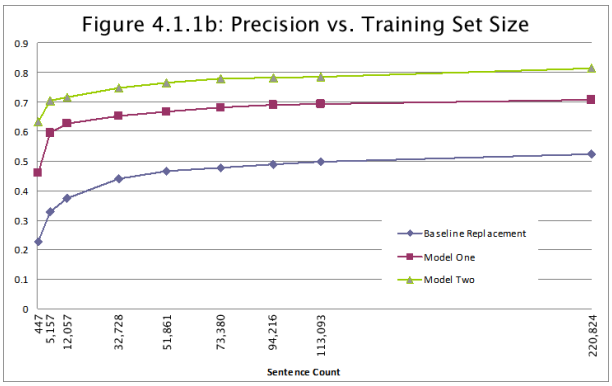
As expected, the AER declines (which means the model is doing better), as the training set size increases. We see the largest drops in AER early on because the model can make good use of the additional data. As the training set size increases, the return from adding additional training data decreases.

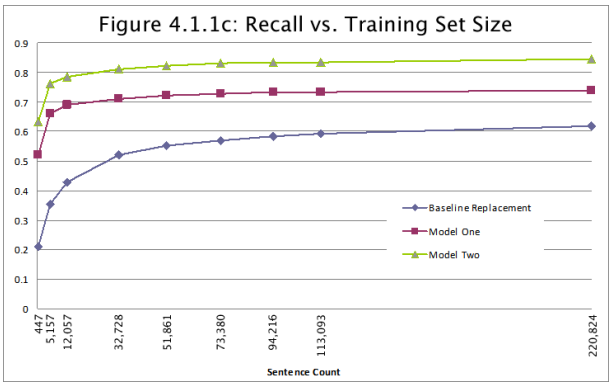
If limited on time, which can be a limitation with these models because they can take a long time to train, then our learning curve shows the

sweet spot to be around 75,000 sentence pairs for Model One and Model Two. Adding additional sentences, such as with our 94k and 113k training sets, barely decreases AER and does not provide enough improvement for the amount of extra training time required. Both models continue to show improvement between the 113k and 221k training sets, but each model's AER only decreases by 0.01, which is not worth the extra time required by training on an additional 108k sentence pairs.

With the exception of the 447 sentence pair set, the difference between Model One and Model Two AER scores remains almost constant. The all other AER scores their difference is generally between 0.095 and 0.106. Model One starts the curve with 0.516 AER on the 447 set while Model Two starts with 0.367 AER. They each end on the 221k set with 0.280 and 0.174 respectively.

Figures 4.1.1b and 4.1.1c show the precision and recall for each model. As is slightly evident in the curves, and is definitely clear in the numbers, the recall quickly levels off for Model One and Model Two as the training size increases while their precision scores level off less quickly and seem to slowly creep upwards.





4.1.2 Alignment Comparison

Figure 4.1.2 is a prime example of the kinds of errors in Model One that Model Two attempts to fix. The figure shows a single French-English sentence pair. Brackets [ ] indicate "sure" truth alignments while parentheses ( ) indicate "possible" truth alignments. The hash marks # indicate the alignments that the models chose. Black hash marks are alignments that both Model One and Model Two chose, red are alignments chosen solely by Model One, and green are alignments chosen only by Model Two.

One frequent problem with Model One is that it does not take any advantage of the fact that language alignments roughly tend to fall along the diagonal since its distortion probabilities are uniform. It makes spurious alignments such as aligning "pas" with "not" in the top right of the figure. Instead, the correct alignment is the word "no" at the beginning of the sentence. Model One makes this error because "pas" is likely to translate as both "not" and "no", and since the distortion probabilities are uniform, it has no incentive to align with the word that is more along the diagonal. These errors can easily be resolved by giving more weight to alignments that lie along the diagonal.

Model Two attempts to make use of this notion with the incorporation of a learned distortion probability. The example in Figure 4.1.2 shows how many of Model One's errors are fixed with Model Two. Instead of aligning "pas" incorrectly in the top right of the figure, Model Two aligns it with the sure alignment of "no". Model Two also correctly aligns the two French words before "pas", and the "pas" that occurs at the end of the French sentence. Model Two is also able to pick up several other sure alignments that Model One aligned to NULL, such as "en" and "la" near the middle of the sentence. Model Two does not fix all problems, as it still makes the same incorrect alignment for "matiere" in the middle of the sentence. Also, while it moves the alignment choice for "exclusivement" closer to the correct alignment, it still gets it wrong.

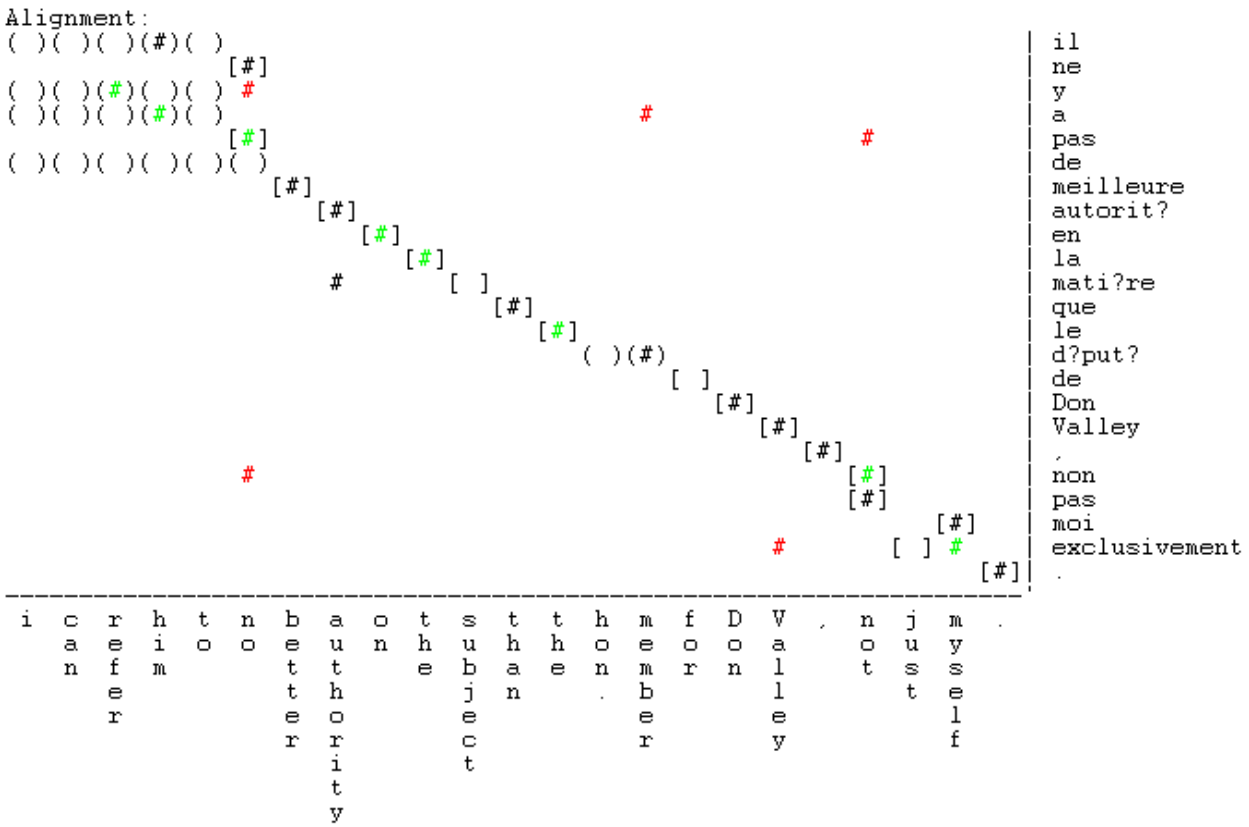


Figure 4.1.2: Brackets [ ] indicate "sure" alignments while parentheses ( ) indicate "possible" alignments. Hash marks # indicate the alignments that the models chose. Black hash marks are alignments that both Model One and Model Two chose, red are alignments only Model One chose, and green are alignments only Model Two chose.

4.2 Translation (Decoder)

We ran the decoder with several different settings and language models. We used two different language models from our previous assignment. The first model, which we will refer to as MLE, is a baseline empirical unigram model that includes a pseudo-count of one for the unknown token. The second model, which we will refer to as LI, is a linearly interpolated model consisting of a Simple Good Turing (SGT) smoothed unigram model, and a bigram and trigram each smoothed with Katz backoff. The bigram model internally includes an SGT unigram model that it backs off to. The trigram model internally contains a bigram with Katz backoff that it backs off to.

For all results we trained the language model using 90,000 sentences and used a separate set of 10,000 sentences as validation data to set the parameters of LinInterp. We trained the alignment models using 80,253 sentence pairs. We tried several values for the lmweight, transweight, and lengthweight parameters of the decoder. We continue this section by first presenting our motivation for trying various decoder parameters. We then finish with WER and BLEU-4 scores for each of our tests.

4.2.1 Translation Examples

In this section, as well as Section 4.2.2 and 4.2.3, we will use an abbreviated naming convention to specify the models and parameters used during a test. We will abbreviate Model One and Model Two and M1 and M2 respectively. We will also specify the lmweight, transweight, and lengthweight parameters as LM, WA, and L respectively. For example, a test using Model Two with the linearly interpolated language model and decoder parameters of 0.4 lmweight, 0.6 transweight, and 1.0 lengthweight will be abbreviated as "M2 LI 0.4LM 0.6WA 1.0L".

Figure 4.2.1 shows the first sentence that the decoder translates. While the alignment model/language model pairs with the default decoder settings of 0.6LM 0.4WA 1.0L successfully translated some of the words, they lack several

important words such as the word "is". To remedy the situation we tried flipping the weights on the language model and alignment model, which produced the favorable translation of M2 LI 0.4LM 0.6WA 1.000.

We also tried leaving the default LM and WA parameters and increasing the length parameter to encourage longer sentences that perhaps would contain more words from the correct translation. This resulted in the bottom four outputs in Figure 4.2.1. The last of these tests, M2 LI 0.6LM 0.4WA 1.500L, produced a longer but much better translation than the other translations. With these encouraging results we decided to run the decoder through the end on all of the test settings in Figure 4.2.1. As we show in the next two sections, while the first translation output in Figure 4.2.1 was encouraging, the default settings ended up producing better translations on other sentences in the set.

```
                French: monsieur le Orateur, ma question se adresse ? le ministre charg? de les transports.
               English: Mr. Speaker, my question is directed to the Minister of Transport.
M1  LI 0.6LM 0.4WA 1.000L: Mr. -- Warren, my question to the Minister of Transport.
M1 MLE 0.6LM 0.4WA 1.000L: Mr. remarks, question the Transport CIDA Minister
M2  LI 0.6LM 0.4WA 1.000L: Mr. -- Warren, my question to the Minister of transportation.
M2 MLE 0.6LM 0.4WA 1.000L: Mr. the, question directed to the of Transport.
M2  LI 0.4LM 0.6WA 1.000L: Mr. the --, my question is directed to the Minister Warren of the transportation.
M2  LI 0.4LM 0.6WA 1.500L: Mr. the --, my question is directed to the Minister of Warren of the transportation.
M2  LI 0.6LM 0.4WA 1.125L: Mr. the --, my question is to the Minister of the Warren transportation.
M2  LI 0.6LM 0.4WA 1.250L: Mr. the --, my question is to the Minister of the Warren transportation.
M2  LI 0.6LM 0.4WA 1.375L: Mr. the --, my question is to the Minister of the Warren transportation.
M2  LI 0.6LM 0.4WA 1.500L: Mr. the --, my question is directed to the Minister of the transportation system.
```

Figure 4.2.1: The first sentence translated by the decoder.

4.2.2 WER

Figure 4.2.2 shows the word error rates for all of our decoder tests. We do not spend much time here discussing the results because the BLEU-4 measure is a much more important measure of machine translation quality.

4.2.3 BLEU-4

Figure 4.2.3a shows the BLEU-4 scores for all of our decoder tests. As the figure shows, Model Two with the LinInterp language model and the default decoder settings receives the highest BLEU-4 score. M1 MLE achieved a score of 0.026 and M2 MLE achieved a score of 0.069. While the deviations from the default decoder settings were promising for the first sentence, they all produced uglier translations for later sentences in the test set. The BLEU-4 scores also show the importance of a good language model, as the jump from MLE to LI in Model One (the two bottom bars) is quite significant.

An interesting insight can be found by looking at the log n-gram scores and brevity penalty for each test. Figure 4.2.3b shows the log n-gram scores for the four aligner/language model pairs using the default decoder settings. As one can see, the M1 LI and M2 LI pairs received almost the exact same log n-gram scores. In fact, the M1 LI configuretion has an ever so slightly better set of log n-gram scores.

This is important because the two configurations had fairly different BLEU-4 scores. M2 LI had a score of 0.139 while the M1 LI had a score of 0.129. The difference in their BLEU-4 score is explained by the M1 LI configuration's brevity penalty of 0.879. All Model Two configurations that used the linearly interpolated language model received no brevity penalty (1.0), while M2 MLE had a penalty of 0.806 and M1 MLE had a penalty of 0.449. Thus while M2 LI and M1 LI both had almost identical log n-gram scores, M2 LI achieved a higher BLEU-4 score by not receiving any brevity penalty.

5 Improvements

We believe that our alignment models would benefit the most from incorporating part-of-speech tags and other linguistic features and by including the fertility modeling of IBM Model 3. As we saw in the actual translation with the decoder, a better language model can also never hurt.

6 Member Contributions

Todd implemented Model Two. Todd an Pavani pair programmed the rest of the assignment. Todd generated, organized, and formatted all results while Pavani selected the examples and created the graphs. Todd wrote Sections 1 through 3 of the report while Pavani wrote the initial version of Section 4 and Todd edited Section 4.

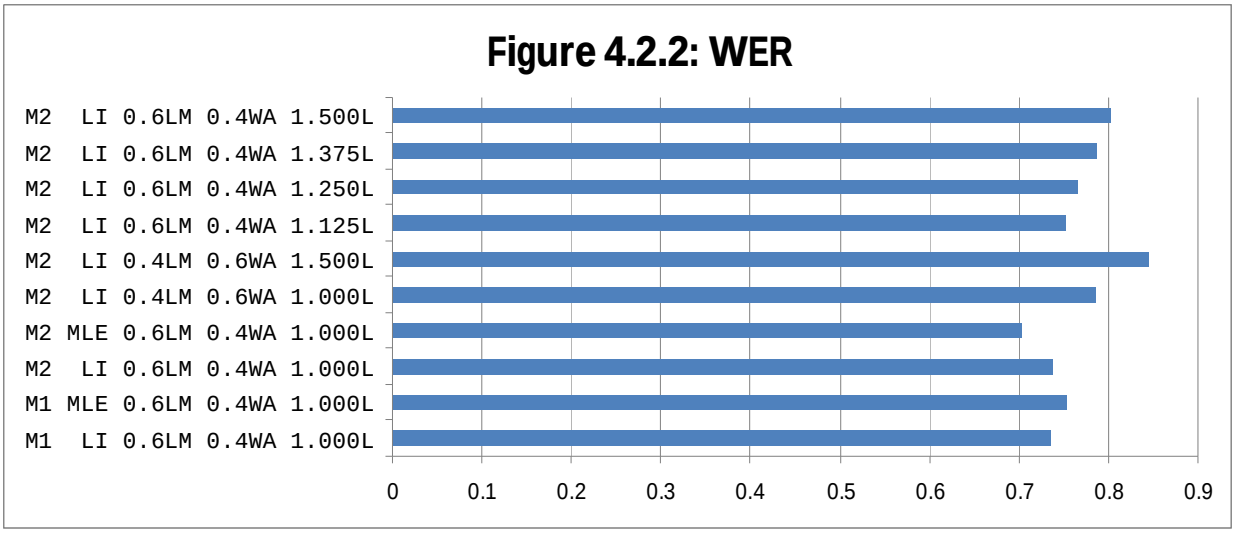


Figure 4.2.2: WER
M2 LI 0.6LM 0.4WA 1.500L
M2 LI 0.6LM 0.4WA 1.375L
M2 LI 0.6LM 0.4WA 1.250L
M2 LI 0.6LM 0.4WA 1.125L
M2 LI 0.4LM 0.6WA 1.500L
M2 LI 0.4LM 0.6WA 1.000L
M2 MLE 0.6LM 0.4WA 1.000L
M2 LI 0.6LM 0.4WA 1.000L
M1 MLE 0.6LM 0.4WA 1.000L
M1 LI 0.6LM 0.4WA 1.000L
0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9

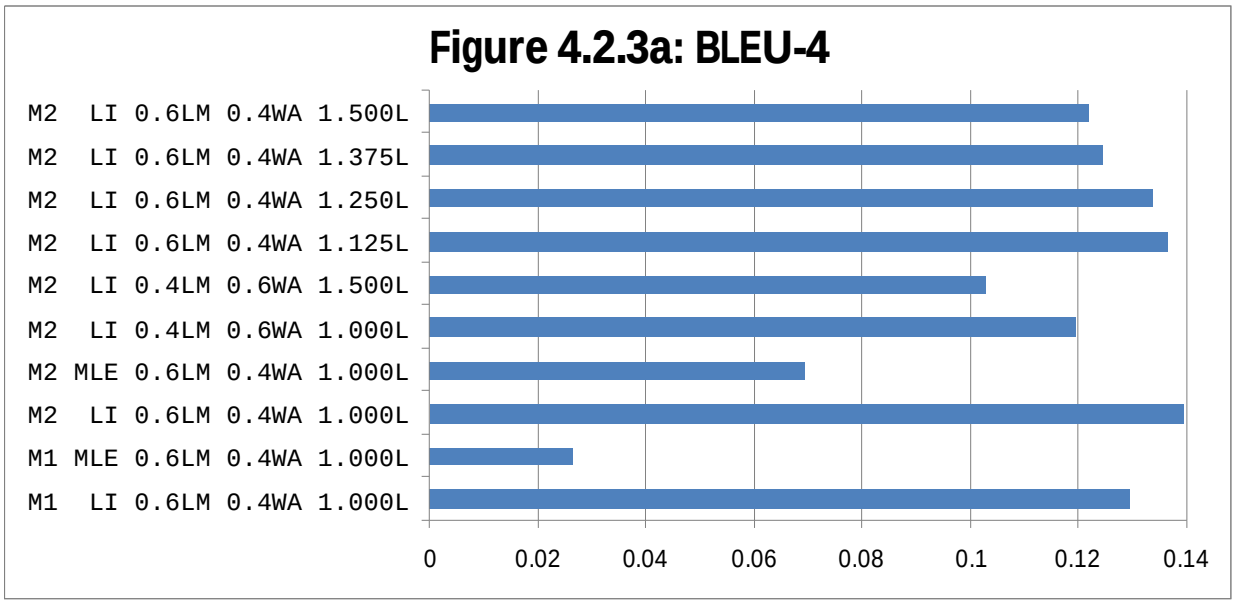


Figure 4.2.3a: BLEU-4
M2 LI 0.6LM 0.4WA 1.500L
M2 LI 0.6LM 0.4WA 1.375L
M2 LI 0.6LM 0.4WA 1.250L
M2 LI 0.6LM 0.4WA 1.125L
M2 LI 0.4LM 0.6WA 1.500L
M2 LI 0.4LM 0.6WA 1.000L
M2 MLE 0.6LM 0.4WA 1.000L
M2 LI 0.6LM 0.4WA 1.000L
M1 MLE 0.6LM 0.4WA 1.000L
M1 LI 0.6LM 0.4WA 1.000L
0 0.02 0.04 0.06 0.08 0.1 0.12 0.14

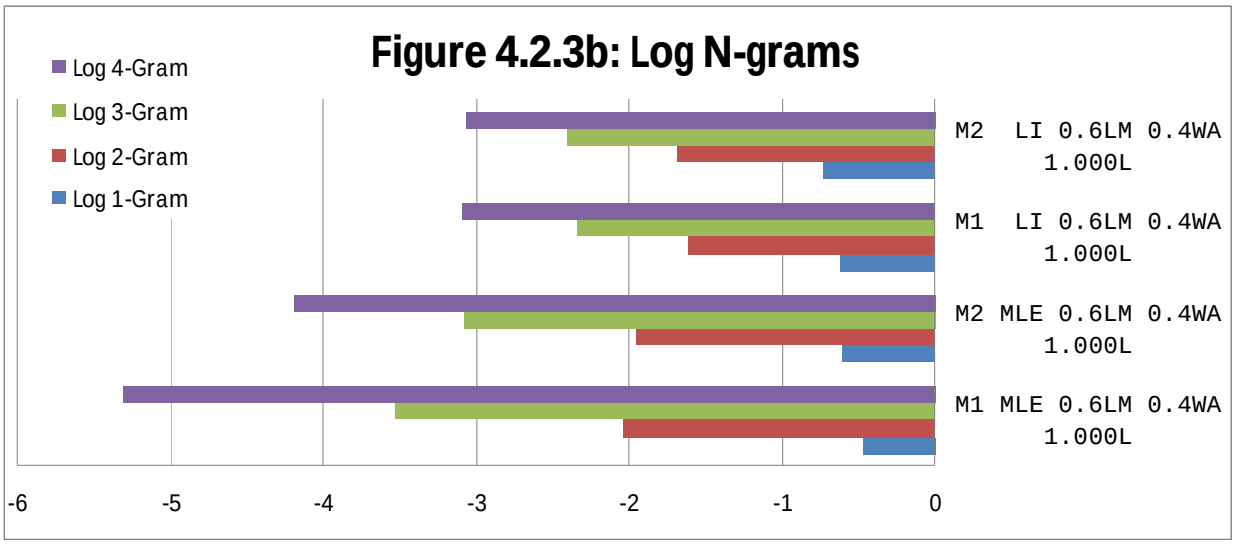


Figure 4.2.3b: Log N-grams
Log 4-Gram
Log 3-Gram
Log 2-Gram
Log 1-Gram
M2 LI 0.6LM 0.4WA 1.000L
M1 LI 0.6LM 0.4WA 1.000L
M2 MLE 0.6LM 0.4WA 1.000L
M1 MLE 0.6LM 0.4WA 1.000L
-6 -5 -4 -3 -2 -1 0