CS 224n Programming Assignment 1: Language Modeling

Todd Sullivan
todd.sullivan@cs.stanford.edu

Pavani Vantimitta
pavani@stanford.edu

1 Introduction

For PA1 we implemented the base unigram, bigram and trigram models. We used Simple Good Turing (SGT) smoothing with simple backing off to lower order models when the conditional distributions did not exist. We also implemented Katz backing off and mixed all of our models in various ways using linear interpolation. Our linear interpolation uses hill-climbing with a random walk to find weights that minimize the test perplexity of a validation set. Our best model based on test perplexity was a linear interpolation between an SGT unigram model, Katz bigram model, and Katz trigram model. When training on the 40,000 sentence corpus, the model has a test perplexity of 181. When training on all of the available data (a little over one million sentences), the model has test perplexity of 107. Our best model based on HUB WER is a linear interpolation between an MLE unigram model, MLE bigram model, and MLE trigram model. The model's best score of 0.05 is achieved when training on a 460,000 sentence corpus.

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.

2.1 LanguageModelTester

We augmented the main method of LanguageModelTester to include extra options such as processing all models in mass and whether to perform a learning curve analysis on the given models. One can also choose whether to use the default weights in the linearly interpolated models or to perform a search that optimizes the weights using validation data and how many threads to use to find the linear interpolation weights of multiple models simultaneously. The first two options allowed us to easily evaluate many models at once and to generate a learning curve for all of our models that includes 49 different training set sizes from 40,000 sentences to one million sentences. The threading option for finding weights from validation data drastically reduced the amount of time required to process our learning curve data since we searched for new optimal weights with each new training set size and the search time totaled at least half of the processing time.

2.2 SuperHelper

To eliminate duplicate computations when evaluating multiple models, we perform all counting, smoothing, and probability distribution creation tasks within a class called SuperHelper. When given training sentences, SuperHelper increments its unigram, bigram, and trigram counters and recalculates the unigram, bigram, and trigram count of counts. This allows us to incrementally add training data when computing our learning curves and eliminates needing to reprocess sentences.

When a model requests its respective probability distribution, the SuperHelper generates the distribution if it has not previously been generated. If the distribution was previously generated, such as when a different model using the same distribution as part of its model previously requested the distribution, SuperHelper simply returns the distribution from the previous calculation. SuperHelper clears all of its probability distributions when it receives new training data. Thus, for example, when evaluating SGTBigram, SGTTrigram, and all of the linearly interpolated models that include Simple Good Turing smoothing, the bigram and trigram counts are smoothed once and the resulting two distributions are used by all of the models.

2.3 Language Model Organization

Each language model implements the LanguageModel interface and has BaseLanguageModel at the root of its inherited ancestors. We modified the original LanguageModel interface to make our task easier. BaseLanguageModel includes all of the common activities among all

models, such as receiving a SuperHelper object, receiving training sentences, and calculating the probability of a sentence.

MLEUnigram, MLEBigram, and MLETrigram directly inherit from BaseLanguageModel. MLETrigram contains MLEBigram, which is used when the two previous words that we are conditioning on have not been seen together. Similarly, MLEBigram contains MLEUnigram, which is used as the probability distribution when there is no conditional distribution for the previous word.

SGTUnigram, SGTBigram, and SGTTrigram are the Simple Good Turing smoothed models described in Gale and Sampson 1995. They inherit from their respective MLE classes. SGTTrigram contains SGTBigram while SGTBigram contains SGTUnigram. To define these classes, we only had to override the getPD function to specify which probability distribution to use and, if applicable, which lower order model to include.

KatzBigram and KatzTrigram inherit from their respective MLE classes. KatzBigram contains SGTUnigram while KatzTrigram contains KatzBigram. If the conditional probability does not exist, each class uses the lower-order model in the same fashion as our other models.

Each linearly interpolated model includes some combination of the previously mentioned models. LinInterpBaseUB and LinInterpBaseUBT inherit from MLEBigram and MLETrigram respectively and include the extra methods required to implement linear interpolation. Each of the actual linearly interpolated models, such as LinInterpSGTUniSGTBiSGTTri which uses all SGT models, only have to override getPD and setDefaultWeights.

2.4 Other Details

We modified Counter to contain a double called zeroItemMass. zeroItemMass is zero by default, and is the value that Counter returns when the key does not exist. We use this for our SGT distributions.

We use several other classes to perform various tasks. FindWeights performs a variant of hill-climbing with random walks to find the optimal set of weights for a linearly interpolated model. Results records various metrics such as processing time and perplexities for each model being evaluated. ProcessLearningRate processes the learning rate output and generates a large CSV file that can be loaded into Excel to create graphs.

3 The Language Models

In this section we prove that our language models contain proper probability distributions. Our MLEUnigram model includes a pseudocount of one for the unknown token while the bigram and trigram counts do not include any pseudocount for the existence on the unknown token in any context. The unknown token is treated as a explicit member of the vocabulary.

We calculate all of our probabilities using the counts of each ngram and a counter that maps counts to the probability that an ngram will have in the distribution given it has the specific count. This map is produced from the count of counts. We create the probability distributions in this way because it was easier when doing Simple Good Turing smoothing.

The probabilities for MLEUnigram, MLEBigram, and MLETrigram are computed by the SuperHelper.empirical method. The SGT models use SuperHelper.simpleGoodTuring. The Katz models use SuperHelper.katzBackoff.

3.1 MLEUnigram

$$\sum_{w \in vocabulary} P_{MLE}(w) = \sum_{w \in vocabulary} \frac{C(w)}{TotalCount}$$

$$= \frac{1}{TotalCount} \sum_{w \in vocabulary} C(w)$$

$$= \frac{1}{TotalCount} (TotalCount) = 1$$

3.2 MLEBigram

For a given context (previous word) w_1,

$$\sum_{w \in vocabulary} P_{MLE}(w \mid w_1) = \sum_{w \in vocabulary} \frac{C(w_1 w)}{TotalCount_{w_1}}$$

$$= \frac{1}{TotalCount_{w_1}} \sum_{w \in vocabulary} C(w_1 w)$$

$$= \frac{1}{TotalCount_{w_1}} (TotalCount_{w_1}) = 1$$

3.3 MLETrigram

For a given context w_1w_2,

$$\sum_{w \in vocabulary} P_{MLE}(w \mid w_1w_2)$$

$$= \sum_{w \in vocabulary} \frac{C(w_1w_2w)}{TotalCount_{w_1w_2}}$$

$$= \frac{1}{TotalCount_{w_1w_2}} \sum_{w \in vocabulary} C(w_1w_2w)$$

$$= \frac{1}{TotalCount_{w_1w_2}} (TotalCount_{w_1w_2}) = 1$$

3.4 SGTUnigram, SGTBigram, SGTTrigram

We calculate the probabilities for all three SGT models using their count of counts data structure. Thus the process is identical for all three. We will show the case of a trigram. For the other two, just change w_1w_2 in all lines to either w_1 or null as appropriate.

Note: COC(0, w_1w_2) is the number of zero count ngrams with w_1w_2 as the first two words and zeroMass$_{w1w2}$ is the amount of zero mass assigned to the condition w_1w_2.

$$\sum_{w \in vocabulary} P_{SGT}(w \mid w_1w_2)$$

$$= \left(\sum_{\substack{C(w_1w_2w)>0, \\ w \in vocabulary}} P_{SGT}(w \mid w_1w_2) \right) + \left(\sum_{\substack{C(w_1w_2w)=0, \\ w \in vocabulary}} P_{SGT}(w \mid w_1w_2) \right)$$

$$= \left(\sum_{\substack{C(w_1w_2w)>0, \\ w \in vocabulary}} (1 - zeroMass_{w_1w_2}) \cdot \frac{C*(w_1w_2w)}{C*TotalCount_{w_1w_2}} \right) + \left(\sum_{\substack{C(w_1w_2w)=0, \\ w \in vocabulary}} \frac{zeroMass_{w_1w_2}}{COC(0, w_1w_2)} \right)$$

$$= \left(\frac{(1 - zeroMass_{w_1w_2})}{C*TotalCount_{w_1w_2}} \sum_{\substack{C(w_1w_2w)>0, \\ w \in vocabulary}} C*(w_1w_2w) \right) + \left(zeroMass_{w_1w_2} \sum_{\substack{C(w_1w_2w)=0, \\ w \in vocabulary}} \frac{1}{COC(0, w_1w_2)} \right)$$

$$= \left(\frac{(1 - zeroMass_{w_1w_2})}{C*TotalCount_{w_1w_2}} C*TotalCount_{w_1w_2} \right) + (zeroMass_{w_1w_2} \cdot 1)$$

$$= 1 - zeroMass_{w_1w_2} + zeroMass_{w_1w_2} = 1$$

3.5 KatzBigram and KatzTrigram

Katz backoff uses SGT probability distributions and gives the zero count mass that would normally be distributed evenly to the zero count events by SGT to a lower order model. We show the case of a trigram. The bigram case is the exact same, except that it uses $P_{SGT}(w)$ for the backoff part. Note: zeroMassSGT is the amount of zero mass assigned to the conditional distribution by SGT.

$$\sum_{w \in vocabulary} P_{Katz}(w \mid w_1w_2)$$

$$= \left(\sum_{\substack{C(w_1w_2w)>0, \\ w \in vocabulary}} P_{SGT}(w \mid w_1w_2) \right) + \left(\alpha(w_1w_2) \sum_{\substack{C(w_1w_2w)=0, \\ w \in vocabulary}} P_{Katz}(w \mid w_2) \right)$$

$$= (1 - zeroMassSGT) + \left(\frac{1 - \sum_{\substack{C(w_1w_2w)>0, \\ w \in vocabulary}} P_{Katz}(w \mid w_1w_2)}{1 - \sum_{\substack{C(w_1w_2w)>0, \\ w \in vocabulary}} P_{Katz}(w \mid w_2)} \sum_{\substack{C(w_1w_2w)=0, \\ w \in vocabulary}} P_{Katz}(w \mid w_2) \right)$$

$$= (1 - zeroMassSGT) + \left(\frac{zeroMassSGT}{\sum_{\substack{C(w_1w_2w)=0, \\ w \in vocabulary}} P_{Katz}(w \mid w_2)} \sum_{\substack{C(w_1w_2w)=0, \\ w \in vocabulary}} P_{Katz}(w \mid w_2) \right)$$

$$= 1 - zeroMass + zeroMass = 1$$

3.6 Linearly Interpolated Models

All linearly interpolated models use weights that sum to one and always use some combination of the previously listed models. We present the case of LinInterpSGTUniSGTBiSGTTri. The other cases are identical.

$$\sum_{w_1w_2w_3} P_{\text{LinInterpSGTUniSGTBi SGTTri}}(w_1w_2w_3; \alpha, \beta, \gamma)$$

$$= \sum_{w_1w_2w_3} \alpha P_{\text{SGT}}(w_3) + \beta P_{\text{SGT}}(w_3 \mid w_2) + \gamma P_{\text{SGT}}(w_3 \mid w_1w_2)$$

$$= \alpha \sum_{w_1w_2w_3} P_{\text{SGT}}(w_3) + \beta \sum_{w_1w_2w_3} P_{\text{SGT}}(w_3 \mid w_2) + \gamma \sum_{w_1w_2w_3} P_{\text{SGT}}(w_3 \mid w_1w_2)$$

$$= \alpha \cdot 1 + \beta \cdot 1 + \gamma \cdot 1 = \alpha + \beta + \gamma = 1$$

4 Performance Analysis

4.1 40,000 Sentence Training Set

4.1.1 Test Set Perplexity

Figure 4.1.1 shows the test set perplexity for most of our models when training on the 40,000 sentence training set. SGTUnigram, MLEUnigram, KatzTrigram, and SGTTrigram are not included because they had large perplexities of 823, 896, 2924, and 8380 respectively. Our best model, LinInterpSGTUniKatzBiKatzTri, has a perplexity of 181. As the figure shows, linearly interpolating between any of the base models is better than the base models themselves. Additionally, interpolating with unigram, bigram, and trigram models is always better than interpolating with unigram and bigram models. Amongst the unigram/bigram interpolations, smoothing the unigram model seems to be more important than smoothing the bigram model.

With the exception of LinInterpSGTUniKatzBiKatzTri, smoothing does not seem to be as important for the uni/bi/tri interpolations. Many of these different combinations of model interpolations give scores that are only different by one or two units. Nevertheless, most of the interpolations with a smoothed unigram model have better performance than without unigram smoothing. All interpolations with at least two smoothed models perform better than the interpolations with only one smoothed model, but there is no clear indication for which model should ideally be smoothed.

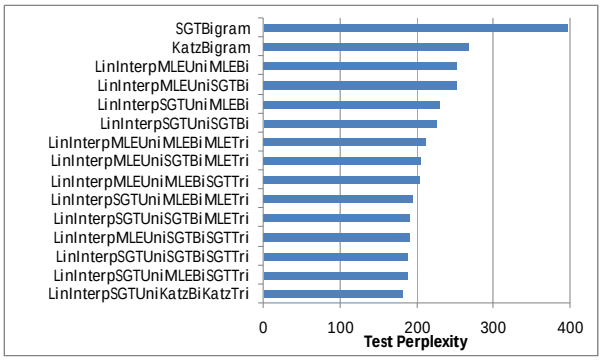

Figure 4.1.1: Test Perplexity when training on the 40,000 sentence training set. A larger version of this figure is on the last page.

4.2 1 Million Sentence Training Set

4.2.1 Test Set Perplexity

Figure 4.2.1 shows the test set perplexity for most of our models when training on the 40,000 sentence training set. Again, SGTUnigram, MLEUnigram, KatzTrigram, and SGTTrigram are not included because they had large perplexities of 628, 895, 906, and 1413 respectively. Our best model, LinInterpSGTUniKatzBiKatzTri, has a perplexity of 107. Similar to the 40,000 sentence training set case, interpolation is always better than each base model alone and uni/bi/tri interpolation is always better than uni/bi interpolation. For uni/bi interpolation, smoothing the unigram model is still more important than smoothing the bigram model. An important note is that while the two uni/bi interpolations with non-smoothed unigram models switched places, their scores are only different starting on the third decimal place.

While LinInterpSGTUniKatzBiKatzTri remains the best uni/bi/tri interpolation model, many of the other models moved around in the performance ranking. The best uni/bi/tri interpolation model has a perplexity of 107.3614 while the worst has a score of 112.4313 – a difference of 5 units between the best and worst. In the uni/bi interpolation case, the best model has a score of 162.9235 and the worst has a score of 166.4266 – a difference of 3.5 units between the best and worst. For the 40,000 sentence training set, the uni/bi/tri difference was 30.7 and the uni/bi difference was 26.2. This shows that as the training set size increases, the use of smoothed models within interpolation becomes less important. This is as expected because with a larger dataset we can trust the counts more.

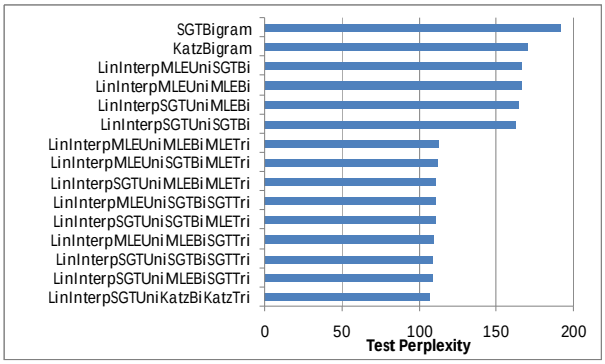

Figure 4.2.1: Test Perplexity when training on the one million sentence training set. A larger version of this figure is on the last page.

4.3 Learning Curves

With the exception of LinInterpSGTUniKatzBiKatzTri, we recorded learning curve data for all of our models. We did not record learning curve data for LinInterpSGTUniKatzBiKatzTri because we created the model later than the other models and did not have time to compute the results. Our learning curve data contains training perplexity, test perplexity, and HUB WER for training set sizes ranging from 40,000 to 1,000,000 in 20,000 sentence increments.

4.3.1 Training Perplexity

As expected for all models, the training perplexity increases as the training set size increases. This is because with more data points our models are less capable of fitting tightly to all data points. Figure 4.3.1 shows the two bands with lowest perplexity. The lowest curve is the MLETrigram while the one immediately above it is SGTTrigram. These two curves are expected to be in this order because the MLETrigram maximizes the likelihood of the training data, and should thus give higher probability to the training set than smoothing the MLETrigram counts. The second band of curves contains all of the uni/bi/tri interpolation models. These models are roughly ordered by the amount of smoothing that occurs in each. The more models that are smoothed in the interpolation, the higher its training perplexity.

4.3.2 Test Perplexity

Figure 4.3.2 shows how using more data improves the performance of all of our models (except the unigram models, which remain fairly constant). We only included the linearly interpolated models because the others have far worse performance and would make the graph unreadable. All curves shown have a decreasing perplexity with a decreasing slope, but none of the curves appear to have leveled out yet. Thus we expect that more training data would continue to improve performance with some significant.

The uni/bi interpolations form the top band with LinInterpMLEUniMLEBi being covered up by the LinInterpMLEUniSGTBi curve, which shows that when not smoothing the MLE unigram model, smoothing the bigram model has little effect. This top band clearly shows that smoothing the unigram model is always better (in terms of perplexity score) than using the MLE unigram model. Additionally, smoothing both is always a little better than only smoothing the unigram model.

The lower band contains all of the uni/bi/tri interpolations. The highest curve in the lower band contains no smoothing on any of the inner models. As we move down to lower curves, the number of inner models that are smoothed gradually increases.

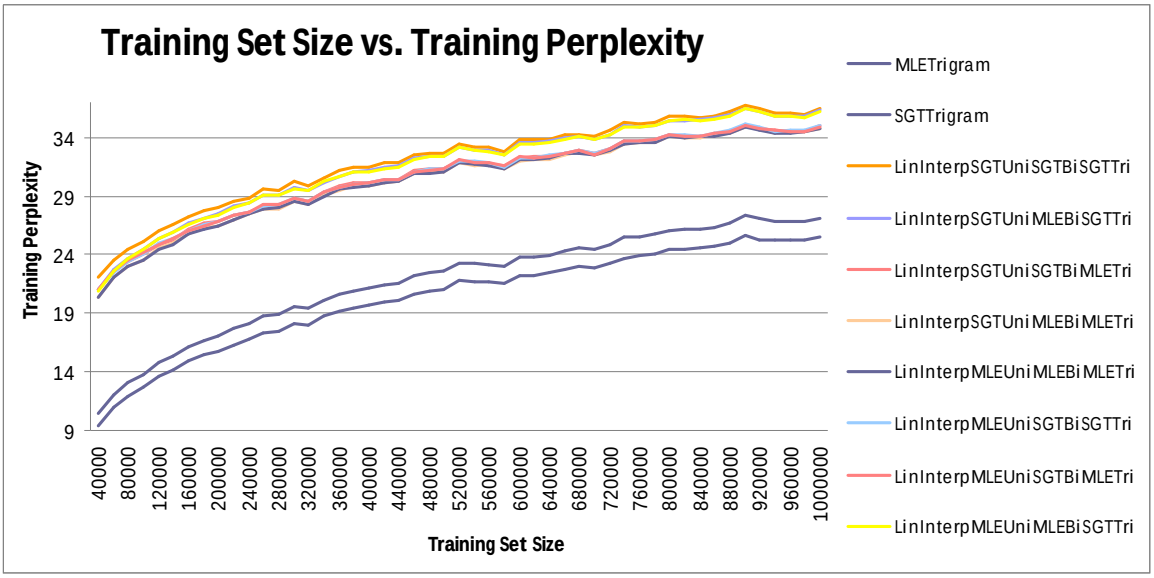

Figure 4.3.1: Training Perplexity Curves.

Additionally, as the training set size increases, the difference between the scores of the best uni/bi interpolation and the worst uni/bi/tri interpolation increases. This suggests that the uni/bi/tri interpolation can make better use of the addition data, which is expected because with more data the inner trigram model becomes more useful. One can also see this in the optimal weights that are found for each model as the training set size increases. As the training set size increases, the amount of weight given to the unigram, and then later bigram, decreases because the trigram becomes more powerful.

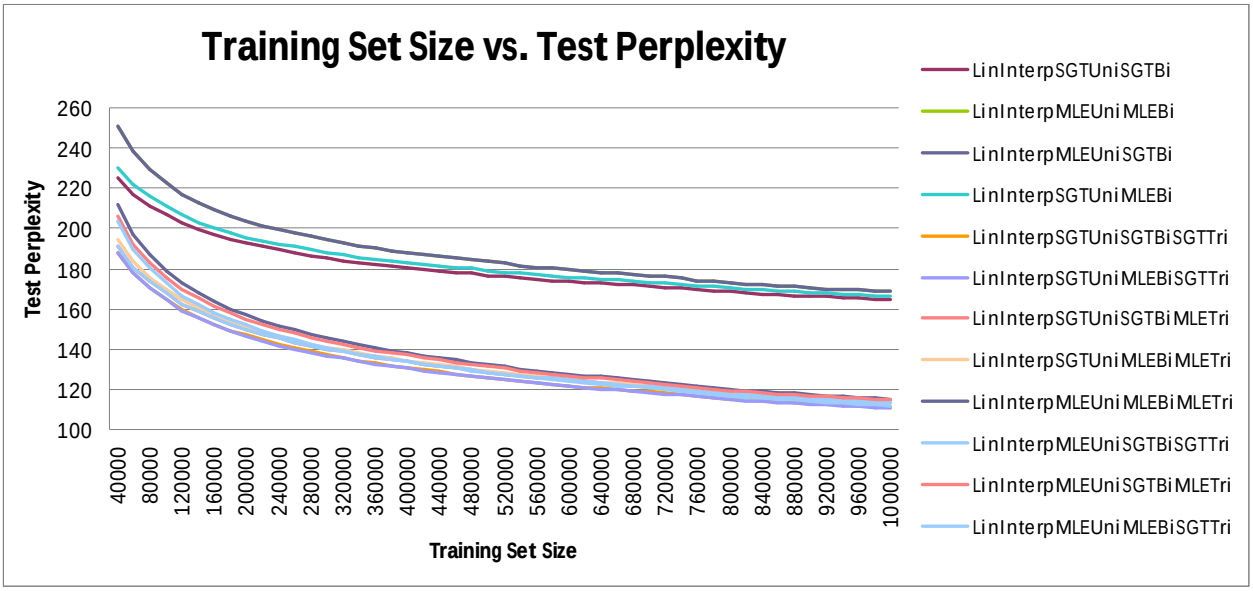

Figure 4.3.2: Test Perplexity Curves.

4.3.3 Trigram Issues

Our base trigram with smoothing, SGTTrigram, is able to almost perfectly fit the training data, but is absolutely pathetic on the test set. One can hardly see the existence of the training perplexity curve because it is between 10 and 30. The test perplexity curve shows a pleasantly steep decrease in perplexity as the training set size increases, but it never even reaches the 900 or so perplexity of the MLE unigram model. We suspect that this is the case because we simply do not have enough data for the trigram model to be used by itself. The trigram model is perfectly capable of overfitting the datasets that we had, but even one million sentences is not enough to obtain accurate counts for each conditional probability distribution.

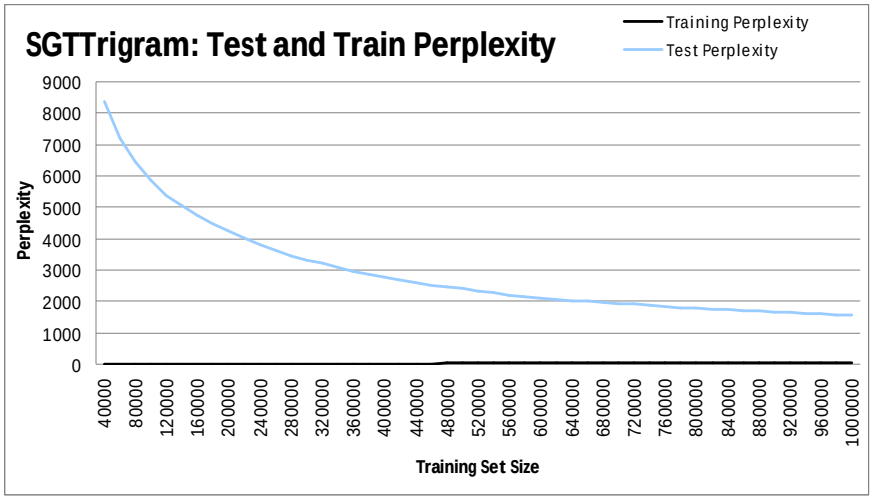

Figure 4.3.4: SGTTrigram Training Perplexity and Test Perplexity Curves.

4.3.4 HUB WER

Figure 4.3.3 shows how erratic the HUB WER is. We obtain our lowest WER of exactly 0.05 with LinInterpMLEUniMLEBiMLETri and LinInterpSGTUniMLEBiMLETri when training on a 460,000 corpus. The majority of the models seem to reach their lowest WER around this point. The sweet spot in terms of training set size seems to be between 380,000 and 540,000 sentences.

The erratic behavior is quite puzzling. Many curves overlap each other and cross over each other at various points. The two models that achieve a 0.05 score overlap the entire time except for the few lime green lines that are seen in sections such as around 740,000. Most of the curves bounce all over the place yet LinInterpSGTUniSGTBi flatlines from 500,000 to 920,000.

Our only guess at why we see all of this strange behavior is that as the training set size increases, the amount of probability mass allotted to zero count elements as well as to the unknown token changes drastically. This causes the jumpy, unpredictable behavior. For most models, the smaller training set sizes do not give a proper allotment to the zero count elements and unknown token. These models hit their sweet spot for these allotments within the range from 380,000 to 540,000 sentences. After that, the allotments become too small, which causes each model's WER to become worse.

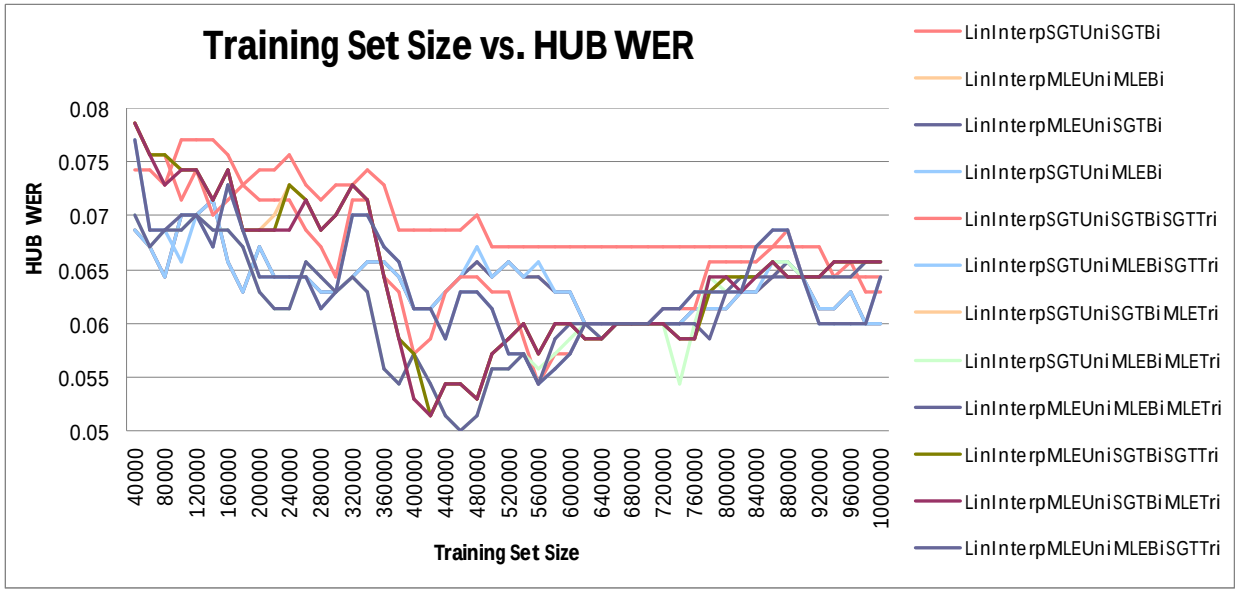

Figure 4.3.3: HUB WER Curves.

4.4 Sentence Generation

In this section we present several random sentences generated by our LinInterpSGTUniKatzBiKatzTriLanguageModel model. Our model performed well when producing short sentences such as "i would like to evaluate them", "this increase is vital in their lives", and "the debate is closed". At the same time, the model generated many sentences that ended abruptly or missed a vital component such as a verb. Some examples are: "the public and social justices" and "i welcome the prospect of the". Additionally, the model produced sentences that were too long and that feel more like multiple poorly written sentences stuck together. One such example is "the court of making it on the hoof of the introduction of the taiwanese nineteen ninety contrary to the resolution we must never again congratulate mr de silguy for their contributions representing various new states". The previous sentence also has other problems that seemed frequent, such as saying "Court of Making It", which matches the often seen template of "____ of ____" where the second blank is not a word that generally should go is such a sequence.

4.5 Speech Recognition

In this section we will cover some issues with the speech recognition problems using our LinInterpSGTUniKatzBiKatzTriLanguageModel model. The most frequent problem seemed to be from the popularity of certain words such as "the". For example, our model chose "the fed spokesman declined to comment as usual" over "a fed spokesman declined to comment as usual". For many of these errors, we cannot think of any changes to our language models that would remedy the situation. We believe that one needs additional context to solve problems such as the fed spokesman lines. Without any additional information such as the previous and next sentences in the speech, many humans would simply be guessing between the two.

Another example of an error that is due to one word being overly abundant is when our model chose "inevitably one child asks how did to get off the tape" over "inevitably one child asks how did you get off the tape". After analyzing the counts, there does not seem to be an easy fix to the situation because the training corpus contains the following counts: "how did to" = 0, "did to get" = 0, "to get off" = 1, "to get" = 114, "to" = 26,060, "how did you" = 0, "did you get" = 0, "you get off" = 0, "did you" = 8, "you get" = 3, "you" = 2346. As one can see, the word "to" appears an order of magnitude more often than "you". The disparity balloons even more when using the one million sentence corpus. Due to these reasons, we believe there is no amount of smoothing or similar tricks that could solve this problem. The only potential improvements can come from other techniques such as incorporating parts of speech or other information beyond counts of ngrams.

5 Member Contributions

Todd implemented the Simple Good Turing smoothing, the linear interpolation weights optimization, and all optimizations involving the SuperHelper. Todd and Pavani pair programmed the rest of the assignment. Todd wrote the entire report and collected all results. Pavani created the graphs.

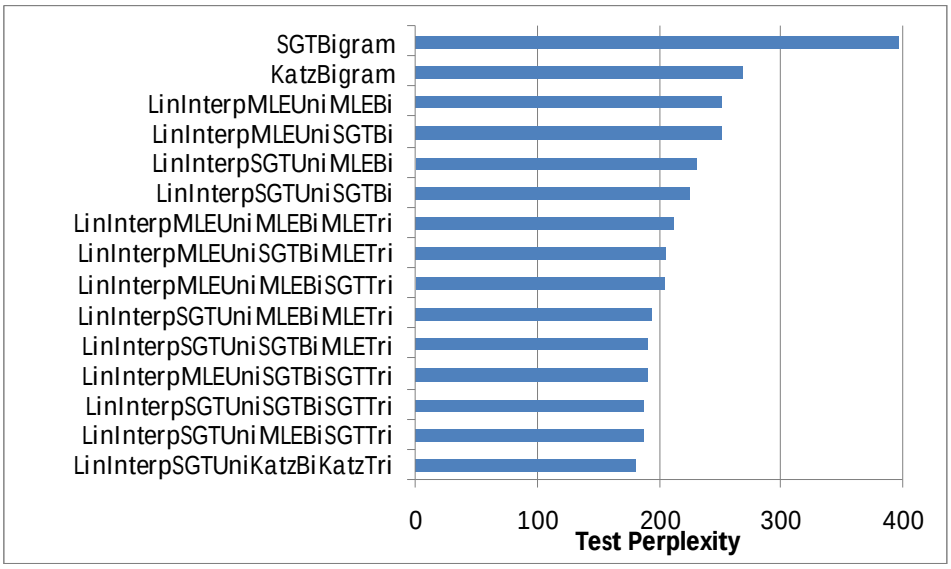

Figure 4.1.1: Test Perplexity when training on the 40,000 sentence training set.

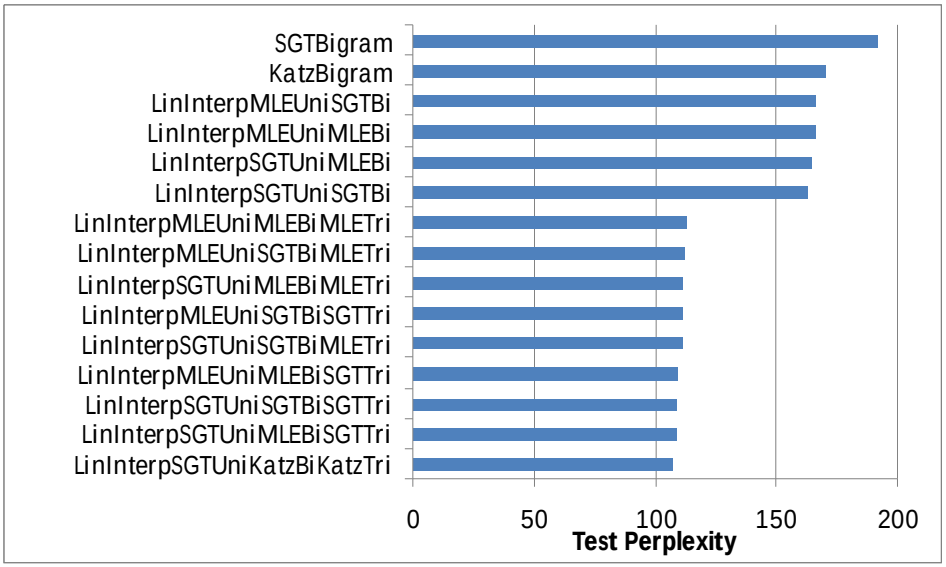

Figure 4.2.1: Test Perplexity when training on the one million sentence training set.