CSE 143: Assignment 1

Jacob Doar jjbagins@ucsc.edu
Daniel Xiong dxiong5@ucsc.edu
Edward Christenson edfchris@ucsc.edu

Due Jan. 26, 2020

1 Introduction

We created n-gram models ($n = 1, 2, 3$) and trained each model on the same subset of the One Billion Word Language Modeling Benchmark dataset. After training the n-gram models, each model's perplexity was calculated on the training, development, and testing data. We then implemented linear interpolation smoothing to try to achieve better results.

2 Tools Used

All code was written in Python 3.8.0

3 Handling Out-of-vocabulary Words

In order to handle OOV words, a dictionary was constructed with every token and its number of occurrences. Using the dictionary of token occurrences, all tokens that were found 3 or more times were placed in a set which gave us the entire vocabulary of our model. From there it was trivial to loop through every token in the training set and convert tokens with less than 3 occurrences to <UNK> symbols. For the development data and test data all tokens that were not found within the vocabulary were replaced with the <UNK> symbol.

4 N-gram Models

4.1 Unigram Model (n=1)

Training the unigram model was relatively simple and was done in a two step process. The simplicity of this model comes from the fact that the probability of each token is independent of previous tokens. The first step involved getting a total token count in the data and also counting the number of times each unique token appears. A dictionary was created, to hold each token and their count, where the keys where each unique token encountered in the data and values where their corresponding frequency in the data. With the dictionary of token frequencies and the total word count, the final step involved creating another dictionary where each key is a unique token and each value is the probability of that token being encountered. The probability of each token was calculated by dividing the frequency of that token over the total token count. This dictionary now holds each token and its probability of occurring and was suitable to be used as the unigram model.

4.2 Bigram/Trigram Models (n=2,3)

In order to efficiently train the bigram and trigram models a two step process was used. The first step involved creating a dictionary whose keys were tuples made up of the previous n-1 words that we referred to as contexts. Each of these contexts is mapped to another dictionary that contained key value pairs made up of words that followed said context in the training data and its number of occurrences. The next step was to create a similar data structure that contained the probability of a word following a given context using the counts from the first step. Using this two step process, a model was constructed that could give the probability of a word following any given context while still being efficient to train and access post training.

4.3 Results

Table 1: Perplexity Scores (rounded to nearest thousandth)

	Unigram	Bigram	Trigram
Training data	976.544	74.278	6.431
Development data	892.247	∞	∞
Test data	896.499	∞	∞

None of the models performed that well on any of the data sets, except for the data they were trained on. Both the bigram and trigram models got a score of ∞ on the development and training data because they were given a word following a context in a configuration they never saw during training and assigned it a probability of zero.

5 Linear Interpolation Smoothing

To make our model work better, we implemented linear interpolation smoothing. Our smoothed model is denoted as θ':

$$\theta'_{x_j|x_{j-1}x_{j-2}} = \lambda_1\theta_{x_j} + \lambda_2\theta_{x_j|x_{j-1}} + \lambda_3\theta_{x_j|x_{j-1}x_{j-2}}$$

where hyper-parameters $\lambda_1, \lambda_2, \lambda_3$ are multiplied to the unigram, bigram, and trigram models respectively.

5.1 Hyper-parameter Experimentation

Table 2: Perplexity Scores (rounded to nearest thousandth) with various hyper-parameters

	$\lambda_1 = 0.1$ $\lambda_2 = 0.3$ $\lambda_3 = 0.6$	$\lambda_1 = 0.7$ $\lambda_2 = 0.15$ $\lambda_3 = 0.15$	$\lambda_1 = 0.15$ $\lambda_2 = 0.7$ $\lambda_3 = 0.15$	$\lambda_1 = 0.15$ $\lambda_2 = 0.15$ $\lambda_3 = 0.7$	$\lambda_1 = 0.33$ $\lambda_2 = 0.33$ $\lambda_3 = 0.33$
Training Data	10.573	34.429	40.090	9.030	18.223
Development Data	1366.801	572.858	1693.706	997.127	769.867
Test Data	1359.333	572.640	1688.903	992.024	767.683

For the hyper-parameters $\lambda_1 = 0.1$, $\lambda_2 = 0.3$, and $\lambda_3 = 0.6$, we observed perplexities of 10.573 and 1366.801 for the training and development data, respectively.

The hyper-parameters $\lambda_1 = 0.7$, $\lambda_2 = 0.15$, and $\lambda_3 = 0.15$ resulted in the lowest development and test data perplexities (572.858 and 572.640, respectively). Hyper-parameter values $\lambda_1 = 0.15$, $\lambda_2 = 0.15$, and $\lambda_3 = 0.7$ resulted in the lowest perplexity score (9.030) for the training data, but it was not as good as just the trigram model alone (6.431), as observed in §5.3, Table 1.

Every experiment using linear interpolation smoothing resulted in models that reported non-∞ perplexity scores on the development and test data, compared to the ∞ perplexity scores observed by bigram and trigram models. This is due to the fact that the unigram will always assign a non-zero probability since it is observing either an <UNK> symbol or a known word within the vocabulary.

6 Training Data Experimentation

Table 3: Perplexity Scores (rounded to nearest thousandth) using half the training data

	Unigram	Bigram	Trigram
Training data	816.490	61.020	5.374
Development data	723.123	∞	∞
Test data	725.551	∞	∞

We trained our models using only half of the training data. For the unigram model, we found that the perplexities on the development and test data were lower than when we trained the model on the full dataset. In the case of the bigram and trigram models, the observed perplexities were still ∞.

This new unigram model had a lower perplexity because during training the model saw fewer unique tokens, meaning the probability of each word occurring increased. Explicitly, given a token x and the total number of unique tokens n, the probability of x occurring, $p(x) = \frac{c_{x_{1:n}}(x)}{n}$, would increase with a smaller n. Higher probabilities for each token would then decrease the perplexity of the model, as they are inversely related.

The bigram and trigram models still reported ∞ because they rely on context. Since we trained these models with half of the training data, they were not able to see as much context, therefore the perplexity would not get any better.

7 OOV Experiemtation

Table 4: Perplexity Scores (rounded to nearest thousandth) with OOV margin < 5

	Unigram	Bigram	Trigram	Smoothed Model λ_1=0.1, λ_2=0.3, λ_3=0.6
Training Data	803.485	73.665	7.134	11.691
Development Data	754.3	∞	∞	1082.713
Test Data	756.689	∞	∞	1076.658

The default margin for words considered OOV was 3 or less occurrences within the training data. As seen in the above table, when the margin was set to 5 or less occurrences the unigram and smoothed models score's improved by a significant factor, while the bigram and trigram models experienced marginal changes to their scores.

The improvement for the unigram comes from it having to learn less words which leads to it giving higher probabilities to the remaining words that it does learn. Since the unigram model is one of the three models that goes into the smoothed model it experiences a similar improvement in score.

Unlike the unigram model, the bigram and trigram models rely on context which means that learning a smaller vocabulary makes less of a difference. Any tokens replaced by <UNK> decrease the size of the vocabulary, but do not change the complexity of the contexts that the bigram and trigram models have to learn. For this reason changing the OOV margin does not significantly effect the perplexity scores of the bigram or trigram models.