NLU Assignment 1 Report

Lokesh (14355)
lokeshn@iisc.ac.in

Abstract

Below presented is the report of the assignment 1 which is about building an Ngram model, calculating perplexity after applying all possible smoothing techniques and then finally generating a sentence from the language model.

1 Introduction

Ngrams are a popular algorithm in NLU to model a language model. N defines the context window the model captures. The performance of a language model is measured by some metrics called perplexity. Lower the value of perplexity the better the model is.

2 Building Language Model

The dataset is split into train and test in the ratio 90:10. I built language model using biGrams and triGrams on the training data. To handle the out-of-vocabulary problem, I treat 10% of the words that occur exactly once in the training set as out of vocabulary words and replace them with ¡UNK¿ token. One more preprocessing that i do is to append ¡starttoken¿ and ¡stoptoken¿ as a prefix and suffix of the sentence respectively.Once these preprocessing is done, all that is left is to count the number of occurences of the different word sequences. I store the word counts in a dictionary. The key of the dictionary is the sequences of words concatenated by space " " and the value being the corresponding counts. All the remaining manipulations are then done using these counts.

2.1 Probability Measure

To calculate the probability, I use kneser-ney probability measure. Then, calculation of perplexity is just an exponent of the summation of the log of keyserney Probabilities. But the issue with this measure is that the recursive formulation given in the text book takes too much time to execute for higher order N-Grams. Hence i am sticking to biGram for perplexity readings. However, if i increase the order to more than 2 Perplexity is guaranteed to come down.

2.2 Observation

One observation i could make is that the perplexity value gets better as i keep increasing the number of UNK tokens. This observation is attributed to the fact that the unknown words in the test set hinder the probability of a sentence very much however well framed the sentence might be. Hence a tendency to increase the number of unknowns would fetch us better perplexity measure. But such a model would fail in sentence generation as many UNK tokens would appear in the generated sentence. Hence i see this as a trade-off. we trade-off the perplexity to sentence generation.

Below mentioned is the table of my observations about the model

unknown words	**Perplexity**
10% words that occur once	225
all words that occue once	137

Table 1: Perplexity in Gutenberg

But i chose to retain the 10% words as unknowns even though i am not getting a good perplexity value as the model might favor good sentence generation. One other way to handle this is to construct two models, one that is used for perplexity measure and the other for sentence generation.

3 Sentence Generation Algorithm

```
import numpy as np
```

```
def geberate_sentence():
  sentence = ""
  used = {}
  iterate triGram
    candidate = find most
                frequent trigram
    used[candidate] = 1
    sentence = sentence + candidate
    """
    Sentence is initialized
    """
    Extend the sentence
    using trigrams
    """
    maximal possible sentence
    constructed using trigrams
    """
    Remove start and stop tokens
    Replace 'unk' with unknown
    words in training set.
    return sentence
```

As described above the sentence generation algorithm tries to formulate the sentence according to the most frequent trigrams. It takes the most frequent trigram and initializes the sentence. It remembers the used trigrams and marks them so that the same sequence does not get repeated in the sentence. From then on, the sentence generator tries to picj the completing trigram as much as possible. A completing trigram is a one that have the first two words as the last two words in the sentence. The generated sentence contains some noise which was introduced by us(starttokens, stoptokens, unk). Hence we need to post process the generated sentence. I remove the start and stop tokens. For unk tokens, I simply replace them randomly with the 10% of the one-frequent words that i set aside as out-of-vocabulary words initially.

4 Output of the Program

Configuration : train = Brown, Test = Brown
train : 46071 test : 11269
sentence is : it is not a man who had been a good deal of pleasure ; ;
****
Configuration : train = Gutenberg, Test = Gutenberg
train : 78850 test : 19702
sentence is : of the lord , and the lord said the little jackal .
****
Configuration : train = Brown + Gutenberg, Test = Brown
train : 46071 test : 11269
sentence is : it is not a man who had been a good deal of attention .
****
Configuration : train = Brown + Gutenberg, Test = Gutenberg
train : 57340 test : 0
sentence is : it is not a single stage we may say that the united states , and the other hand , the first time in the world .
****

Corpus	Perp	Sentence
Brown + Brown	289	it is not a man who had been a good deal of attention.
Gutenberg + Gutenberg	221	of the lord , and the lord said the little jackal .
Gutenberg + Brown, Brown	290	it is not a single stage we may say that the united states , and the other hand , the first time in the world .
Gutenberg + Brown, Gutenberg	276	it is not a man who had been a good deal of attention

Table 2: Summary of Results