E1-246 Assignment 1

G P Shrivatsa Bhargav
SR 14865
bhargavs@iisc.ac.in

1 Tasks

1.1 Task1

Divide each dataset into train, dev, and test (splits are up to you). Let the training splits be D1-Train and D2-Train. Implement and build the best LM in the following settings and evaluate.

- S1: Train: D1-Train, Test: D1-Test
- S2: Train: D2-Train, Test: D2-Test
- S3: Train: D1-Train + D2-Train, Test: D1-Test
- S4: Train: D1-Train + D2-Train, Test: D2-Test

1.2 Task 2

Using the best model, you will also be required to generate a sentence of 10 tokens. Include a script with the name generate_sentence.sh which when executed will generate a sentence as described above.

2 Preprocessing

We use two datasets namely Brown and Gutenberg. Each dataset is divided into training,validation and testing sets. The dataset for each scenario S1,S2,S3,S4 is prepared accordingly. The sizes of the splits are in table 1. The data is split randomly at document level so that the model doesn't see any text from the test data - similar to real world setting.

Other preprocessing steps are :

- Removing annotations from the brown corpus
- Removing punctuation
- Removing numbers
- Removing unwanted newlines,spaces and tabs (if there are multiple consecutive occurrences)
- Replacing words occurring fewer than 3 times with <unk>
- Adding <pad> $order - 1$ times to the beginning of the data.

Setting	Training+Validation	Validation	Testing
S1	904672	90623	100416
S2	1825798	1001831	276748
S3	2730470	1092454	100,416
S4	2730470	1092454	276,784

Table 1: Dataset sizes (number of words)

3 Models

Bi-gram Kneser Ney smoothing has been implemented according to(Jurafsky and Martin, 2009). The probability of a bigram is given by:

$$p_{KN}(w_i|w_{i-1}) = \frac{max(c(w_{i-1}, w_i) - D, 0)}{c(w_{i-1})} + \gamma(w_{i-1}) p_{cont}(w_i)$$

$$p_{cont}(w) = \frac{|w_{i-1}|c(w_{i-1}, w) > 0|}{|(w_i, w_j)|c(w_i, w_j) > 0|}$$

$$\gamma(w_{i-1}) = \frac{D}{c(w_{i-1})}|w|c(w_{i-1}, w) > 0|$$

$$perplexity = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{p(w_i|w_{i-1})}}$$

The value of D was tuned on validation set. D=0.85 was found to be optimal. The performance of the model was evaluated using perplexity.

3.1 Results

The perplexities for bigram Kneser Ney are in table 2.
The brown corpus is a collection of many small documents written by many different authors. So the model can't capture the different writing styles effectively. The Gutenberg corpus is double the size of Brown.But it is collection of a few large documents written by a few authors. Due to more data and low writing style diversity , the model performs better on Gutenberg corpus. In S3 and S4, the size of the corpus appears to be playing a role. Gutenberg overshadows Brown in S3 and S4 so the perplexity on Gutenberg test set is lower(S4) than Brown(S3)

Setting	Perplexity
S1	423
S2	317
S3	550
S4	346

Table 2: Perplexities

4 Sentence generation

Sentence generation uses the bigram kneser ney model to evaluate probable continuations after each generated token. Algorithm 1 is used to generate sentences.

```
Result: Sentence of k tokens.
initialize sentence $s = n - 1$ <pad> tokens
  where $n$ is the order of the model;
while $length\ of\ s < k$ do
    1. pick a word $w$ with uniform probability
       from the vocabulary;
    2. Form an ngram $ng$ using the last $n - 1$
       tokens of $s$ and $w$;
    3. Pick a number $a$ from $[0, 1]$ with
       uniform probability;
    if $P(a < ng)$ then
        Append $w$ to $s$;
    end
end
```

Algorithm 1: Sentence generation

Example sentences:

- **S1**: chicago onions and the most important that he hung golden
- **S2**: meremoth nature of air that believeth in the beasts band
- **S3**: rebels most amiable shall carry over a natural concession there

As a bigram model is used, the words of the sentences make sense in windows of size 2.

5 Other models

A general n-gram modified Kneser Ney model(Chen and Goodman, 1996) was implemented. There were many corner cases that made both the ngram probability and interpolated n-1 gram probabilities zero. In other cases, when a particular context is never seen, it is not clear how to handle the division by zero. Straightforward approaches would result in a "score" for the input but not "probabilities". This makes it hard to compare models because of the different metrics. 4-gram modified gave a score of 1300 for S1.

6 Source code

The source code for the mentioned models can be found at https://github.com/gpsbhargav/LanguageModels

References

Stanley F. Chen and Joshua Goodman. 1996. An empirical study of smoothing techniques for language modeling. In *Proceedings of the 34th Annual Meeting on Association for Computational Linguistics*, ACL '96, pages 310–318, Stroudsburg, PA, USA. Association for Computational Linguistics.

Daniel Jurafsky and James H. Martin. 2009. *Speech and Language Processing (2Nd Edition)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.