**Assignment 1**

---

**1.2 N-gram Language Modeling & Perplexity (28%)**

---

**(a)**

## Building N-gram Models

I built the N-gram models in several key steps:

1. **Data Preprocessing:** Each line in the dataset are treated as a sequence. The sequence was tokenized using whitespace as the delimiter. Special tokens were added for the start (START) and end (STOP) of sequences.
2. **Vocabulary Construction:** The vocabulary was built from the training data, with rare words (appearing less than three times) replaced by a special 'UNK' token. The final vocabulary size for the training set was 26,602.
3. **Unigram Model:** The unigram model was constructed by counting the frequency of each token in the training data. This model did not incorporate START tokens, as they do not influence the probability of unigrams. Each token's probability was calculated based on its frequency relative to the total number of tokens.
4. **Bigram Model:** The bigram model involved calculating the frequencies of pairs of consecutive tokens (bigrams). This model included bigrams starting with the START token to capture the beginning of sentences. Probabilities of bigrams were computed as the frequency of the bigram divided by the frequency of its first token.
5. **Trigram Model:** Similarly, the trigram model was developed by calculating frequencies of triplets of consecutive tokens. It included trigrams that began with one or two START tokens, representing the beginning of sentences. The probability of a trigram was calculated as the frequency of the trigram divided by the frequency of its first two tokens.
6. **Special Handling of START and STOP Tokens:** The START token was specially handled in the bigram and trigram models. While it did not form part of the vocabulary (as it is not predicted), it was used as a contextual token at the beginning of sentences. The STOP token was included in all models to signify the end of sentences.

**Model Sizes:**

| Model Type | Number of N-grams |
|---|---|
| Unigram | 26,602 |
| Bigram | 510,391 |
| Trigram | 1,116,160 |

**(b)**

## Computing Perplexity without Smoothing

Perplexity without smoothing was calculated using the following approach:

$$PPL(S) = \exp\left(-\frac{1}{|S|}\sum_{i=1}^{|S|}\log P(s_i|s_1,...,s_{i-1})\right) \quad (1)$$

Where:

- $PPL(S)$ is the perplexity of sequence $S$.
- $P(s_i|s_1,...,s_{i-1})$ is the probability of token $s_i$ given its previous tokens $s_1,...,s_{i-1}$.

The final perplexity is calculated by taking the weighted average over all sequences (discounted by the number of words in each line):
np.average(a_list_of_perplexities, weights=a_list_of_lengths).

**(c)**

## Potential Downsides of Laplace Smoothing

Laplace smoothing disproportionately inflates the probabilities of rare n-grams by adding 1 to each count. Since probabilities has to add up to 1, this effectively reduces the probabilities of more frequent n-grams, which might bias the model towards rare n-grams.

**Empirical Evidence:** Comparing the perplexity scores of the training set with and without Laplace smoothing, we can see that the perplexity of the unigram model is relatively stable, but a significant increase is observed for the bigram and trigram models.

| Model Type | Training Set Perplexity Unsmoothed | Training Set Perplexity Smoothed |
|---|---|---|
| Unigram | 857.98 | 856.69 |
| Bigram | 69.34 | 1268.06 |
| Trigram | 6.36 | 4073.12 |

**(d)**

## Effect of Different Laplace Smoothing Factors

When $k$ is small, model gives a high perplexity due to underestimating the probability of unseen N-grams. As $k$ increases, the model starts giving lower perplexities, when values of $k$ approaching 1 the model starts to increase perplexity again.
Seems like there's a non-linear relationship between $k$ and perplexity, with an optimal range where the perplexity is minimized. I believe this is related to bias and variance tradeoff, where we need to use empirical methods to find the optimal value of $k$.

**(e)**

## Perplexity Scores of N-gram Models

The perplexity scores for unigram, bigram, and trigram models were calculated both with and without Laplace smoothing. These scores describe the performance of the models in predicting the sequences in the dataset.

**Perplexity without Smoothing:**

| Model Type | Training Set | Development Set | Test Set |
|---|---|---|---|
| Unigram | 857.98 | 786.03 | 790.21 |
| Bigram | 69.34 | Infinity | Infinity |
| Trigram | 6.36 | Infinity | Infinity |

In the bigram and trigram models, when evaluated on the validation and test datasets, the perplexity reached an infinite value. This occurred due to the probability of some bigram/trigrams being zero. Given that no smoothing technique was applied in this instance, it is reasonable for the perplexity to be infinite.

**Perplexity with Laplace Smoothing:**

| Model Type | Training Set | Development Set | Test Set |
|---|---|---|---|
| Unigram | 856.69 | 786.26 | 790.28 |
| Bigram | 1268.06 | 1460.34 | 1459.36 |
| Trigram | 4073.12 | 6389.98 | 6367.86 |

With Laplace smoothing, the perplexity of the unigram model remains relatively stable, but a significant increase is observed for the bigram and trigram models. It shows that Laplace smoothing resolves issues with unseen N-grams but, at the same time, may introduce inaccuracies in the estimation of N-gram probabilities for higher-order models. Here we might be oversmoothing the bigram and trigram models, which gives to a much higher perplexity score after smoothing.

### 1.3 Interpolation (17%)

**(a)**

#### Finding Optimal Combinations of $\lambda_1, \lambda_2, \lambda_3$ for Interpolation

The goal was to minimize perplexity, which means a better predictive model.

**Experiment Results:**

| $\lambda_1$ | $\lambda_2$ | $\lambda_3$ | Training Perplexity | Dev Perplexity | Test Perplexity |
|---|---|---|---|---|---|
| 0.33 | 0.33 | 0.34 | 15.97 | 373.73 | 376.34 |
| 0.2 | 0.3 | 0.5 | 12.07 | 440.87 | 444.20 |
| 0.2 | 0.2 | 0.6 | 10.79 | 495.22 | 498.82 |
| 0.1 | 0.15 | 0.75 | 9.09 | 689.24 | 695.13 |
| 0.05 | 0.08 | 0.87 | 8.16 | 1095.98 | 1107.03 |
| 0.1 | 0.3 | 0.6 | 10.45 | 550.59 | 555.71 |

The strategy is gradually shifting the weight from unigram towards bigram and trigram models. Since the weight on higher-order n-grams increased, we see a general trend of reduced training perplexity but development and test perplexity increased. It suggests that higher-order n-grams are more predictive on training data, but they may overfit unseen data.

**(b)**

#### Test Set Perplexity with Optimal Hyperparameters

The optimal combination of hyperparameters are chosen based on their lowest perplexity on the development set.

**Optimal Hyperparameters:**

- $\lambda_1 = 0.33$
- $\lambda_2 = 0.33$
- $\lambda_3 = 0.34$

**Perplexity on Test Set:** 376.34

**(c)**

## Impact of Reduced Training Data on Perplexity

With half the training data, I would expect the perplexity to decrease. Because in my implementation, I replace unseen words with 'UNK' tokens, when we reduce the training data, we would expect to see a lot more 'UNK' tokens which gives a more general model (less variety in vocabulary). When evaluating on the dev and test set we would have a lower perplexity.

**Empirical Evidence:** In the experiment, I deleted half of the training data. We can see compare to the perplexities in part (a), the perplexities on the dev and test sets decreased.

| $\lambda_1$ | $\lambda_2$ | $\lambda_3$ | Training Perplexity | Dev Perplexity | Test Perplexity |
|---|---|---|---|---|---|
| 0.33 | 0.33 | 0.34 | 13.37 | 352.93 | 354.31 |
| 0.2 | 0.3 | 0.5 | 10.09 | 424.28 | 425.62 |
| 0.2 | 0.2 | 0.6 | 9.01 | 477.04 | 478.78 |
| 0.1 | 0.15 | 0.75 | 7.58 | 677.62 | 679.66 |
| 0.05 | 0.08 | 0.87 | 6.81 | 1096.53 | 1099.77 |
| 0.1 | 0.3 | 0.6 | 8.74 | 541.30 | 542.30 |

**(d)**

## Effect of Different UNK Thresholds on Perplexity

Increasing this threshold (from tokens appearing three times to those appearing less than five times) increases the number of 'UNK' tokens, making the model more general and potentially less sensitive to changes in the language.

**Empirical Evidence:** In my experiments, increasing the 'UNK' threshold to 5 led to a decrease in perplexity across, dev and test sets (compare to part(a)).

| $\lambda_1$ | $\lambda_2$ | $\lambda_3$ | Training Perplexity | Dev Perplexity | Test Perplexity |
|---|---|---|---|---|---|
| 0.33 | 0.33 | 0.34 | 17.25 | 306.39 | 308.06 |
| 0.2 | 0.3 | 0.5 | 13.15 | 354.99 | 357.06 |
| 0.2 | 0.2 | 0.6 | 11.80 | 398.27 | 400.44 |
| 0.1 | 0.15 | 0.75 | 9.99 | 542.14 | 545.59 |
| 0.05 | 0.08 | 0.87 | 9.00 | 844.17 | 850.43 |
| 0.1 | 0.3 | 0.6 | 11.43 | 433.26 | 436.39 |

## 2 Byte-Pair Encoding (BPE) (30%)

**(a)**

**(b)**

**(c)**

## 3 WordPiece (25%)

**(a)**

**(b)**

**(c)**

## 4 Open-ended Exploration for Tokenization (10% for 517, extra credit for 447)

**(a)**