Homework 2 NLP 201: N-gram Language Modeling

Soren Larsen

November 2024

1 Introduction

The goal of this assignment is to build and evaluate unigram, bigram, and trigram language models using the provided dataset, which is a subset of the One Billion Word Language Modeling Benchmark. This report outlines the implementation and results for the n-gram models.

2 Data Preprocessing

For training the n-gram models, I utilized the provided training data file `1b_benchmark.train.tokens`, which was processed as follows:

- Tokenization was performed by splitting each line into whitespace-separated tokens.
- Special tokens `<START>` and `<STOP>` were added depending on the n-gram type (e.g., two `<START>` tokens for trigram models and one `<START>` token for bigram models).
- A preprocessing step was employed to count word frequencies. Words occurring less than three times were replaced with the special `<UNK>` token.

3 Model Implementation

The `Ngram` class was created to encapsulate the n-gram modeling logic, including tokenization, preprocessing, and n-gram counting. Below are the main features of the implementation:

3.1 Ngram Model Class

The `Ngram` class includes:

- `tokenize_and_prepare(data)`: This method reads and tokenizes the input data, adding appropriate `<START>` and `<STOP>` tokens.

- `preprocess_and_count(preprocessed_sentences)`: This method counts word frequencies and replaces words occurring less than three times with `<UNK>`.
- `count_ngrams()`: This method generates and counts n-grams from the tokenized sentences.

3.1.1 Handling Edge Cases in Short Sequences

To ensure consistent handling of edge cases, every sentence is tokenized with `<START>` and `<STOP>` tokens added at the beginning and end, respectively. This preprocessing step ensures that even short sentences, which might not naturally contain enough tokens for higher-order n-grams, are padded to create complete n-grams. For instance, a single-token sentence like "`dog`" would be transformed into "`<START> dog <STOP>`" for a bigram model or "`<START> <START> dog <STOP>`" for a trigram model. This approach guarantees robust n-gram construction and avoids issues with missing context during training and evaluation.

3.2 Trade-offs in N-Gram Models

Higher-order n-grams, such as trigrams, improve contextual accuracy but present challenges:

- **Data Sparsity**: As the n-gram order increases, the number of unique n-grams grows exponentially. This sparsity may lead to zero probabilities for unseen sequences, particularly in smaller datasets.
- **Overfitting**: Trigram models can overfit to the training data, memorizing specific patterns that do not generalize well to unseen data.
- **Computational Costs**: Higher-order n-grams require more storage and computational resources to store and evaluate probabilities, making them less efficient compared to unigrams or bigrams.

Despite these challenges, smoothing techniques such as linear interpolation can alleviate issues related to data sparsity, and proper hyperparameter tuning can help mitigate overfitting.

4 Command Line Interface

A command-line interface is provided to specify the type of n-gram model (unigram, bigram, or trigram). The user specifies the desired n-gram type as a command-line argument when running the program. The implementation handles validation of input and sets the n-gram order accordingly.

5 Results

The implemented models were tested with the following results:

- For the unigram model, there were **80,660 unique n-grams** and a total n-gram count of **1,622,905**.
- For the bigram model, there were **615,645 unique n-grams** and a total n-gram count of **1,622,905**.
- For the trigram model, there were **1,195,986 unique n-grams** and a total n-gram count of **1,622,905**.

These results reflect the progressive increase in the number of unique n-grams as the n-gram order increases. The total n-gram count remains consistent across models, corresponding to the total number of tokens processed.

6 Perplexity Evaluation

The perplexity of a language model is a measure of how well it predicts a sample. Lower perplexity indicates better predictive performance. In this section, I evaluate the perplexity scores for the unigram, bigram, and trigram models on the training, development, and test datasets using Maximum Likelihood Estimation (MLE).

6.1 Perplexity Calculation Example

To illustrate the development and testing process, a quick perplexity calculation was performed on the phrase:

`<START> HDTV . <STOP>`

The results for this example are as follows:

Model	Perplexity on "<START> HDTV . <STOP>"
Unigram	**658.045**
Bigram	**63.708**
Trigram	**39.479**

Table 1: Perplexity results for the phrase "<START> HDTV . <STOP>" across different models.

This example highlights how each model's contextual knowledge influences its predictions. The unigram model assigns a higher perplexity due to its lack of context, while the bigram and trigram models better capture the dependencies within the short phrase.

6.2 Overall Perplexity Results

The perplexity scores for each model and dataset are shown below:

Model	Training Set Perplexity	Development Set Perplexity	Test Set Perplexity
Unigram	**976.544**	**892.247**	**896.499**
Bigram	**55.105**	**19.333**	**19.432**
Trigram	**6.519**	**2.405**	**2.412**

Table 2: Perplexity scores for unigram, bigram, and trigram models on different datasets.

6.3 Discussion of Results

- **Unigram Model**: The unigram model assigns probabilities based solely on individual word frequencies without considering context. This approach results in the highest perplexity values across all datasets, with scores of 976.544 on the training set, 892.247 on the development set, and 896.499 on the test set. These high perplexity values reflect the model's inability to capture contextual dependencies, as it treats each word independently.
- **Bigram Model**: By incorporating the preceding word as context, the bigram model achieves a significant reduction in perplexity compared to the unigram model, with values of 55.105 on the training set, 19.333 on the

development set, and 19.432 on the test set. This demonstrates the impact of modeling word pairs and capturing limited context, which greatly improves predictive accuracy. However, the noticeable difference between the training and development/test perplexity scores suggests that the model still faces challenges when generalizing to new data.

- **Trigram Model**: The trigram model further improves predictive performance by incorporating two preceding words as context. This leads to perplexity values of 6.519 on the training set, 2.405 on the development set, and 2.412 on the test set. The dramatic decrease in perplexity highlights the model's ability to capture complex dependencies within the language. Additionally, the close similarity in development and test set perplexities indicates strong generalization and robust predictive capability, reflecting a well-balanced model complexity.

- **Comparison Across Datasets**: The trend of decreasing perplexity from unigram to trigram models demonstrates the importance of context in language modeling. The unigram model's poor performance, due to its lack of contextual awareness, contrasts sharply with the bigram and trigram models, which show increasingly better predictive capabilities as more context is incorporated. The relatively stable perplexity values between the development and test sets for the trigram model suggest it effectively handles unseen data without overfitting.

- **Overall Observations**: These results confirm that higher-order n-grams capture more context, leading to better predictions and lower perplexity. However, there is a trade-off between model complexity and data sparsity. While the trigram model performs best, it may face data sparsity issues in smaller datasets. The use of smoothing techniques, though not detailed here, would further enhance model robustness by mitigating zero-probability events.

7 Linear Interpolation Smoothing

In this section, I describe the implementation of linear interpolation smoothing for our n-gram language models. The smoothed parameter is defined as:

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

where θ' represents the smoothed parameters and $\lambda_1, \lambda_2, \lambda_3$ are weights on the unsmoothed MLE unigram, bigram, and trigram language models respectively, subject to the constraint $\lambda_1 + \lambda_2 + \lambda_3 = 1$. No additional smoothing is applied to the unigram, bigram, and trigram models being interpolated.

7.1 Debugging Example

To help debug the implementation, I used $\lambda_1 = 0.1, \lambda_2 = 0.3, \lambda_3 = 0.6$ for the evaluation set "HDTV .". The expected perplexity for this example is 48.1. Our calculated perplexity was:

$$\text{Perplexity for "HDTV ."} = 48.114$$

This result serves as a validation point for the correctness of our implementation.

7.2 Specified Hyperparameter Combination

To meet the requirement, I evaluated the model using the specified values $\lambda_1 = 0.3, \lambda_2 = 0.3, \lambda_3 = 0.4$. The corresponding perplexity scores on the training and development datasets are shown below:

λ_1	λ_2	λ_3	Perplexity
0.3	0.3	0.4	**Training: 12.512, Development: 325.475**

Table 3: Perplexity scores on training and development sets for the specified λ values.

7.3 Hyperparameter Tuning

I explored several combinations of λ values to optimize performance using the development data. The selected combinations and their respective perplexity scores are shown below.

λ_1	λ_2	λ_3	Training Perplexity	Development Perplexity
0.1	0.3	0.6	9.138	438.884
0.5	0.3	0.2	20.745	302.701
0.2	0.4	0.4	12.116	336.631
0.25	0.35	0.4	12.304	328.285
0.4	0.4	0.2	19.714	294.843

Table 4: Perplexity scores on training and development sets for different λ combinations.

7.4 Hyperparameter Tuning Combination Rationale

In this section, I provide an in-depth analysis of the five hyperparameter combinations explored during tuning. Each choice reflects different levels of emphasis on unigram, bigram, and trigram models, aiming to optimize performance on the development dataset. These combinations are detailed in Table 4, with the following rationale and observations:

- $\lambda_1 = 0.1, \lambda_2 = 0.3, \lambda_3 = 0.6$**:**
  - *Rationale:* This combination heavily emphasizes the trigram model (60%) while maintaining a moderate weight on bigrams and a minimal fallback to unigrams. The intention here was to maximize the use of contextual information provided by trigrams, which are known to significantly improve perplexity when data sparsity is not an issue.
  - *Outcome:* While this configuration resulted in the lowest training perplexity (9.138), it had the highest development perplexity (438.884). This discrepancy suggests overfitting, where the model learned specific patterns in the training data but struggled to generalize to unseen data.
- $\lambda_1 = 0.5, \lambda_2 = 0.3, \lambda_3 = 0.2$**:**
  - *Rationale:* Here, the unigram model receives the highest weight (50%), reducing the influence of contextual models (bigrams and trigrams). This choice prioritizes individual word probabilities, which can be helpful in sparse data scenarios where contextual n-grams are less reliable.
  - *Outcome:* The development perplexity dropped significantly (302.701) compared to the first combination, reflecting improved generalization. However, the training perplexity increased to 20.745, indicating that the model relied less on specific patterns and more on general trends, potentially underutilizing available context.
- $\lambda_1 = 0.2, \lambda_2 = 0.4, \lambda_3 = 0.4$**:**
  - *Rationale:* This combination balances the contributions of bigram and trigram models while reducing the unigram weight. The intention was to leverage both local and extended contexts equally while retaining a fallback for sparsity through the unigram component.
  - *Outcome:* The training perplexity (12.116) and development perplexity (336.631) showed a balance between overfitting and generalization. This combination highlighted the value of giving equal importance to medium- and high-context models, though it still fell short of achieving optimal development perplexity.
- $\lambda_1 = 0.25, \lambda_2 = 0.35, \lambda_3 = 0.4$**:**
  - *Rationale:* By slightly increasing the weight of the bigram model and reducing the unigram contribution compared to the previous combination, this configuration sought to capture more local contextual patterns without over-relying on the unigram fallback.
  - *Outcome:* The development perplexity improved further (328.285), while the training perplexity remained stable (12.304). This indicates better generalization, as the model avoided overfitting while capturing a wider range of dependencies.

- $\lambda_1 = 0.4, \lambda_2 = 0.4, \lambda_3 = 0.2$**:**
  - *Rationale:* The chosen combination reflects a shift towards unigram and bigram models, with equal emphasis on both and a reduced contribution from the trigram model. This approach minimizes overfitting risks by limiting the reliance on deeper context, which is more prone to sparsity issues.
  - *Outcome:* This configuration yielded the best development perplexity (294.843), with a reasonable training perplexity (19.714). The results suggest that the balance between unigram and bigram components provided robust predictions, while the reduced trigram weight avoided overfitting to high-order n-grams.

Implications of Each Combination: The tuning process highlighted several key insights:

- High λ_3 values (e.g., in Combination 1) excelled at capturing context but were prone to overfitting, especially when training data was sparse or noisy.
- High λ_1 values (e.g., in Combination 2) prioritized generalization but sacrificed context, resulting in higher training perplexity.
- Balanced λ_2 and λ_3 values (e.g., in Combinations 3 and 4) demonstrated strong performance by effectively combining local and extended contextual information.
- Reduced λ_3 weights (e.g., in Combination 5) mitigated overfitting and proved to generalize best to unseen data.

By exploring these configurations, I identified a sweet spot in the balance of unigram, bigram, and trigram contributions. The optimal choice of $\lambda_1 = 0.4$, $\lambda_2 = 0.4$, $\lambda_3 = 0.2$ reflects this balance, achieving the lowest perplexity on the development dataset.

7.5 Best Hyperparameter Selection

Based on the results from the development set, I chose the following λ values as the optimal configuration:

$$\lambda_1 = 0.4, \quad \lambda_2 = 0.4, \quad \lambda_3 = 0.2$$

The perplexity on the test set using these parameters is reported below:

Dataset	**Perplexity**
Test Set	294.279

Table 5: Test set perplexity using optimal λ values.

From this point forward, all remaining experiments were conducted using the interpolated n-gram model, optimized with the best-fit hyperparameters.

7.6 Impact of Reduced Training Data

To evaluate the impact of reducing the training data by half, I measured the perplexity on the previously unseen data. Results are summarized below:

Dataset (Half Training Data)	Perplexity
Development Set	287.385
Test Set	287.604

Table 6: Perplexity scores on unseen data using half of the training data.

7.7

7.7.1 Impact of Reduced Training Data

When evaluating the impact of reducing the training data by half, I observed perplexity scores of 287.385 on the development set and 287.604 on the test set, as shown in Table 6. Compared to the perplexity scores achieved with the full training data, the results suggest an interesting trend:

- **Generalization Improvement**: Reducing the amount of training data can sometimes force the model to rely less on memorization and more on generalizable patterns in the data. In this case, the slight increase in perplexity compared to the test set perplexity of 294.279 using full training data suggests that the model may have been slightly overfitting when trained on the entire dataset. By reducing the training data, the model's ability to generalize appears to have improved, leading to a lower perplexity on unseen data.
- **Trade-Off with Model Complexity**: While reducing the training data can improve generalization, it also risks underfitting if there is insufficient data to learn meaningful patterns. In this scenario, however, the performance remained strong, indicating that the n-gram models were able to retain sufficient contextual information even with the reduced training data.
- **Potential for Data Sparsity Issues**: Higher-order n-gram models like trigrams are particularly sensitive to data sparsity. Reducing the training data could have exacerbated sparsity issues, but the impact was relatively modest here, suggesting that the interpolation method effectively smoothed probabilities and mitigated this risk.

Overall, the results indicate that halving the training data led to a modest decrease in perplexity on unseen data, suggesting that the model's generalization capabilities were improved slightly by limiting overfitting to specific training examples.

7.8 Impact of <UNK> Token Threshold

I compared two approaches for handling rare words: converting all words that appear fewer than 5 times to <UNK> and converting only words that appear once. The perplexities generated use the full training data set again. The results are shown below:

Threshold for <UNK>	Development Perplexity	Test Perplexity
Less than 5 occurrences	258.074	257.557
Only 1 occurrence	323.012	322.529

Table 7: Perplexity scores for different <UNK> token thresholds.

7.8.1 Impact of <UNK> Token Threshold

The results of the experiment, shown in Table 7, highlight the influence of different thresholds for converting rare words to the <UNK> token on model perplexity. Specifically, two thresholds were evaluated: converting all words that appeared fewer than 5 times and converting only words that appeared once. The findings are summarized in 7.

Key Observations:

- **Impact on Perplexity**: Using a lower threshold (fewer than 5 occurrences) to replace words with <UNK> resulted in significantly lower perplexity scores for both the development and test sets compared to the threshold that only converted words appearing once. This suggests that increasing the number of words treated as out-of-vocabulary (OOV) improves the model's generalization capability by reducing overfitting to specific rare words that may not generalize well to unseen data.

- **Generalization Improvement**: The lower perplexity for the "fewer than 5 occurrences" threshold indicates that by treating a larger set of rare words as <UNK>, the model learns more robust probability estimates for previously unseen data. This is because grouping more rare words under the common <UNK> label reduces the likelihood of encountering zero-probability scenarios and helps the model better generalize.

- **Increased Perplexity for Higher Threshold**: Conversely, using a threshold that converts words that appeared just once led to higher perplexity. This is likely due to the model assigning higher weights to words

that appear infrequently, thereby overfitting to those specific words and reducing its ability to predict new or unseen sequences effectively.

- **Balance Between Specificity and Robustness**: Lowering the threshold too much, however, risks overly generalizing the model, where too many distinct words are treated as `<UNK>`. The key takeaway here is to find a balance that maintains sufficient specificity for commonly occurring words while appropriately handling rare words as OOV.

In conclusion, converting all words appearing fewer than 5 times to `<UNK>` improves model perplexity by enhancing generalization capabilities and reducing overfitting to rare tokens. This approach leads to more stable and robust probability estimates compared to a more restrictive threshold that focuses only on the rarest words.