Report

Name: Sami Hammoud

ID: 318461753

E-Mail: H.Sami@campus.technion.ac.il

The tasks:

1. In my quest to find a suitable text corpus for implementing language models, I encountered several challenges. Initially, I attempted to work with large text corpora, such as those available on the http://mattmahoney.net/dc/textdata.html website and Wikipedia sentences dataset (Taken from Kaggle site) and UsTwitterData (mentioned as "fileIdBigData" and "fileIdSmallData" in the notebook both token from source found on google). However, I faced difficulties due to their immense size and the lack of repetitive words within them. When I tried using small portions of these datasets or smaller text files, the results were not satisfactory since there wasn't enough repetitive words on them and this caused in bad probabilities and higher usage of Ram.

To overcome these challenges, I decided to explore alternative datasets that would strike a balance between having enough text for reasonable language models and ensuring computational efficiency. I diligently documented the different datasets I explored, noting their respective IDs for future reference in my notebook (You can see in the second cell in the notebook, and just change the ID in the link – each id set in a parameter – and you will change the data that the notebook works on).

Ultimately, I made the decision to use the "One Fish, Two Fish, Red Fish, Blue Fish" children's book by Dr. Seuss as my chosen text corpus. Although it was relatively small in size, I found it sufficient for testing my code and implementing the language models. Regrettably, due to the limited RAM available in my Google Colab environment, I was unable to run larger datasets that I had prepared.

To summarize, my journey in finding a suitable text corpus involved a process of trial and error. I experimented with various datasets of different sizes and compositions. Ultimately, I made the decision to work with the "One Fish, Two Fish, Red Fish, Blue Fish" book as it provided an appropriate amount of text for my language models. However, the constraints of my computational resources prevented me from utilizing larger datasets. Despite these limitations, I made the best use of the available data to achieve meaningful results.

2. The perplexity of the unigram, bigram and trigram based on the basic implementation of the n-gram model are:
Test perplexity - unigram: 81.992
Test perplexity - bigram: 3.046
Test perplexity - trigram: 1.249

3. The perplexity of the unigram, bigram and trigram based on the modified n-gram, including the delta smoothing modification and the Knesser-Nay smoothing techniques are:
Add-delta smoothing with delta = 1:
Test perplexity - unigram: 81.739
Test perplexity - bigram: 79.678
Test perplexity - trigram: 13.293
Add-delta smoothing with delta = 2:
Test perplexity - unigram: 83.611
Test perplexity - bigram: 94.465
Test perplexity - trigram: 14.299

Knesser-Nay smoothing:
Test perplexity - unigram: 1.112
Test perplexity - bigram: 0.610
Test perplexity - trigram: 4.339

4. The unigram, bigram, and trigram language models were implemented, and their perplexity values were calculated. The perplexity serves as a measure of how well the language model predicts unseen data.

The unigram model, which considers each word as an independent event, has the highest perplexity value of 81.992. This is expected since it lacks any contextual information and treats each word in isolation.

The bigram model, incorporating the previous word as context, significantly improves the perplexity to 3.046. By considering word pairs, it captures some local dependencies and provides better predictions compared to the unigram model.

The trigram model, extending the context to the previous two words, achieves an even lower perplexity of 1.249. It captures more complex dependencies and exhibits better predictive performance compared to the unigram and bigram models.

To address the issue of unseen n-grams and to smooth the probability estimates, two modifications were applied to the trigram model: add-delta smoothing and Kneser-Nay smoothing.

With add-delta smoothing, using delta=1, the perplexity values slightly improved compared to the unsmoothed trigram model. However, the improvement is relatively small, indicating that the impact of smoothing is limited.

Using Kneser-Nay smoothing, the perplexity values showed a more significant improvement. The perplexity decreased to 1.112 for the unigram model, 0.610 for the bigram model, and 4.339 for the trigram model. Kneser-Nay smoothing effectively addresses the issue of unseen n-grams and assigns probability mass to previously unseen word sequences, resulting in better perplexity values.

In terms of advantages and disadvantages, the unigram model is computationally efficient but lacks context and fails to capture dependencies. The bigram model improves upon the unigram model by considering immediate word context but still has limitations in capturing longer-range dependencies. The trigram model further improves the predictive performance by considering a wider context.

Smoothing techniques like add-delta and Kneser-Nay help address the issue of unseen n-grams and improve the perplexity values. However, add-delta smoothing has a limited impact, while Kneser-Nay smoothing demonstrates more substantial improvements.

Overall, the trigram model with Kneser-Nay smoothing achieves the best perplexity values and provides more accurate predictions compared to the other models. It captures local and long-range dependencies, effectively handles unseen n-grams, and yields better results in terms of language modeling.

It is worth noting that the choice of dataset size and composition, as well as other factors such as the tokenization method and the specific implementation details, can also affect the performance and outcomes of the language models. These factors should be considered when interpreting the results and evaluating the models.

- I shall note that the addition of the `if prob:` condition in the `probability` function serves the purpose of mitigating potential numerical stability issues. In language modeling, it is not uncommon to encounter extremely small probabilities, which can be close to zero due to the vast number of possible word combinations. When these probabilities are multiplied together to calculate the score, numerical precision limitations can arise, leading to underflow issues. By introducing the `if prob:` condition, only non-zero probabilities are considered in the score calculation, effectively excluding the cases where the probability is zero. This approach helps to avoid excessive accumulation of extremely small probabilities, improving numerical stability and ensuring reliable computation in the language model.

Mathematical definition for all the models I've implemented:

1. Unigram Language Model:

   - Given a sequence of words W = {w_1, w_2, ..., w_n}, the unigram language model calculates the probability of each word independently without considering any context.
   - The probability of a unigram is calculated as: P(w) = Count(w) / N where Count(w) is the number of occurrences of word w in the training data, and N is the total number of words in the training data.

2. Bigram Language Model:

   - Given a sequence of words W = {w_1, w_2, ..., w_n}, the bigram language model calculates the probability of a word w_i given its preceding word w_{i-1}.
   - The probability of a bigram is calculated as: P(w_i | w_{i-1}) = Count(w_{i-1}, w_i) / Count(w_{i-1}) where Count(w_{i-1}, w_i) is the number of occurrences of the bigram (w_{i-1}, w_i) in the training data, and Count(w_{i-1}) is the number of occurrences of the preceding word w_{i-1} in the training data.

3. Trigram Language Model:

   - Given a sequence of words W = {w_1, w_2, ..., w_n}, the trigram language model calculates the probability of a word w_i given its preceding two words w_{i-2} and w_{i-1}.
   - The probability of a trigram is calculated as: P(w_i | w_{i-2}, w_{i-1}) = Count(w_{i-2}, w_{i-1}, w_i) / Count(w_{i-2}, w_{i-1}) where Count(w_{i-2}, w_{i-1}, w_i) is the number of occurrences of the trigram (w_{i-2}, w_{i-1}, w_i) in the training data, and Count(w_{i-2}, w_{i-1}) is the number of occurrences of the preceding two words (w_{i-2}, w_{i-1}) in the training data.

4. Add-Delta Smoothing: The add-delta smoothing formula is as follows:

   $$P_{add-delta}(w_i \,|w_{i-1}) = (count(w_{i-1}, w_i) + delta) \,/\, (count(w_{i-1}) + delta * V)$$

   In the formula:

   - $count(w_{i-1}, w_i)$ represents the count of the n-gram (w_{i-1}, w_i) in the training data.
   - $count(w_{i-1})$ represents the count of the (n-1)-gram w_{i-1} in the training data.
   - V represents the vocabulary size, i.e., the total number of unique words in the training data.
   - delta is a small constant typically set to 1, but it can be adjusted based on the specific requirements. (in our case we used 1 and 2)

5. Kneser-Ney Smoothing: The Kneser-Ney smoothing formula involves two main steps:

   Discounting: $P_{discounted}(w_i \,|w_{i-1}) = (count(w_{i-1}, w_i) - D) \,/\, count(w_{i-1})$

   In the formula:

- $count(w_{i-1}, w_i)$ represents the count of the n-gram (w_{i-1}, w_i) in the training data.
- $count(w_{i-1})$ represents the count of the (n-1)-gram w_{i-1} in the training data.
- D is the discount value, which is typically computed as $D = max(count(w_{i-1}, w_i) - threshold, 0) / count(w_{i-1})$.

Backoff: $P_{backoff}(w_i \ |w_{i-1}) = lambda(w_{i-1}) * P\_discounted(w_i)$

In the formula:

- $lambda(w_{i-1})$ is a normalization factor that ensures the probabilities sum to 1 over all possible w_i.
- $P_{discounted}(w_i)$ is the discounted probability of the unigram w_i .

For summary:

1. Data Collection and Preparation:
    - The initial step was to choose a text corpus in one language out of English, Hebrew, or Arabic.
    - Several datasets were explored, including large text files and Wikipedia articles.
    - Considerations were made regarding dataset size, composition, and availability of repeatable words.
    - Ultimately, the "One Fish, Two Fish, Red Fish, Blue Fish" children's book by Dr. Seuss was selected as the dataset.
2. Implementation of Language Models:
    - Unigram, bigram, and trigram language models were implemented from scratch, without using existing implementations.
    - The models were developed using the n-gram approach, where n represents the number of consecutive words considered in the model.
    - Probability calculations were performed using maximum likelihood estimation based on the counts of n-grams in the training data.
3. Experimental Methodology:
    - The implemented language models were evaluated using the perplexity metric.
    - Perplexity measures how well the language model predicts unseen data.
    - The test dataset, consisting of a sequence of words, was used to calculate perplexity for each model.
4. Experiment Results:
    - The perplexity values for the unigram, bigram, and trigram language models were reported without any smoothing.
    - The perplexity values were also calculated after applying add-delta smoothing with different delta values and Kneser-Ney smoothing.
    - The perplexity values were compared across the different models and smoothing techniques.
5. Discussion of Results:
    - The unigram model had the highest perplexity, indicating poor prediction performance due to the lack of context.
    - The bigram model performed better than the unigram model, considering the immediately preceding word as context.
    - The trigram model further improved the prediction accuracy by considering the two preceding words as context.
    - Add-delta smoothing improved the perplexity values, but higher delta values led to less effective smoothing.
    - Kneser-Ney smoothing achieved the lowest perplexity values, indicating its effectiveness in capturing long-range dependencies.

The overall analysis and discussion of the results focused on the advantages and disadvantages of each model and smoothing technique. The report highlighted how the models with more context (bigram and trigram) performed better than the unigram model. Additionally, the impact of different smoothing techniques on perplexity values was discussed, with Kneser-Ney smoothing yielding the best results. The report also emphasized the importance of choosing an appropriate dataset and considering computational efficiency while implementing language models.

References:

1. Lab 2-1 solution on the webcourse.

2. Kaggle site for WikiSentences data.

3. Jurfasky & Martin book.

4. Github for finding some data.