Problem Set: Building N-Gram Language Models in PyTorch

May 10, 2023

Problem 1: Unigram Language Model (5 points)

Implement a unigram language model in PyTorch. Given a corpus of text, your model should compute the probability of each word occurring in the text. The probability of a word w_i in a unigram model is given by the equation:

$$P(c_1, c_2, ..., c_n) \approx \prod_{i=1}^{n} P(c_i)$$

Train your model on the given corpus and evaluate its performance using perplexity, which is defined as:

$$Perplexity = 2^{-\frac{1}{N}\sum_{i=1}^{N} \log_2 P(c_i)}$$

Problem 2: N-gram Language Model (10 points)

Implement an n-gram language model in PyTorch. Your model should predict the probability of a word given the previous (n-1) words. The probability of a word c_i given the previous (n-1) words is given by the equation:

$$P(c_1, c_2, ..., c_n) \approx \prod_{i=1}^{n} P(c_i | c_{i-1}, ...c_{i-N})$$

Train your model on the given corpus and evaluate its performance using perplexity. Compare the performance of your n-gram model with the unigram model from Problem 1.

Problem 3: Every-gram Language Model using Linear Interpolation (20 points)

Implement an every-gram language model in PyTorch using linear interpolation. Your model should combine unigram, bigram, trigram, and higher-order n-gram models, where the weights for each n-gram model are computed using linear interpolation. The interpolated probability of a word c_i given the previous (n-1) words is given by the equation:

$$\begin{aligned} P(c_1, c_2, \ldots, c_n) &\approx P(c_i | c_{i-(n-1)}, c_{i-(n-2)}, \ldots, c_{i-1}) \quad (1) \\ &= \lambda_1 P(c_i) + \lambda_2 P(c_i | c_{i-1}) \\ &\quad + \lambda_3 P(c_i | c_{i-2}, c_{i-1}) \\ &\quad + \cdots \\ &\quad + \lambda_n P(c_i | c_{i-(n-1)}, \ldots, c_{i-1}) \quad (2) \end{aligned}$$

where $\lambda_1, \lambda_2, \ldots, \lambda_n$ are the interpolation weights, subject to the constraint:

$$\lambda_1 + \lambda_2 + \lambda_3 + \cdots + \lambda_n = 1$$

Train your model on the given corpus and evaluate its performance using perplexity. Compare the performance of your every-gram model with the unigram and n-gram models from Problems 1 and 2.