Ronak Mehta
Instructor: Noah A. Smith
CSE 517: Natural Language Processing
Winter 2021
Assignment A2 (Coding)
Due: January 29, 2021

1 Problem Statement

We would like to learn a language model p, which is a probability distribution over sentences in $\mathcal{V}^*$, i.e. lists of words from a vocabulary $\mathcal{V}$. Such a distribution should be amenable to sampling and evaluation on new sentences.

2 Dataset

The dataset used is a subset of the One Billion Language Modeling Benchmark. The dataset is pre-split into training, dev, and test sets.

Preprocessing The preprocessing steps on this data were to:

- Split the training, dev, and test sequences into sentences delimited by new lines.
- Tokenize each sentence, delimited by whitespace.
- Append two start tokens `<START>` at the beginning of each sentence, and one stop token `STOP` token at the end. This allows the bigram and trigram model to be well-defined (see Section 3).
- Replace the tokens in the training set that appear less than 3 times to the special unknown token `<UNK>`, and replacing all of these tokens as well as out-of-vocabulary (OOV) tokens in the dev and tests sets to `<UNK>` as well.

3 Model

An ensemble of autoregressive n-gram models were considered to fit this distribution. That is, the probability of a sentence $(w_1, ..., w_T)$ is given by

$$p(w_1, ..., w_T) = \prod_{t=1}^{T} p(w_1 \mid w_{t-1}, w_{t-2}, ..., w_{t-(n-1)}),$$

where w_t for $t < 1$ denotes a `START` token. In particular, we consider an interpolation of a unigram, bigram, and trigram models, parametrized by $\lambda_1, \lambda_2, \lambda_3 \geq 0$ with $\sum_{j=1}^{3} \lambda_j = 1$.

$$p(w_1, ..., w_T) = \prod_{t=1}^{T} \left[\lambda_1 \cdot p(w_t \mid w_{t-1}) + \lambda_2 \cdot p(w_t) + \lambda_3 \cdot p(w_t \mid w_{t-1}, w_{t-2})\right]$$

Implementation Details The maximum likelihood estimate of the above probabilities were stored in a tree data structure. Branches are indexed by words, and a node at depth n contains the count of the n-gram given by the path from the root to that node. For example, for the tree

$$\texttt{root} \overset{\text{“}\texttt{the}\text{”}}{\rightarrow} \texttt{node1} \overset{\text{“}\texttt{cat}\text{”}}{\rightarrow} \texttt{node2},$$

`node2` will contain the count of the bigram (“the”, “cat”) in the training set. The probability vector $p(w \mid \text{“}\texttt{the}\text{”})$ is taken by collecting the counts of all the children nodes of `node1` and normalizing, which is evaluated lazily at inference or generation time. If additive smoothing is applied, then the smoothing parameter α is added to all counts before normalization. $\alpha = 0.0003$ seemed to be a good choice after hyperparameter search, so that value is used in the following experiments.

4 Results

Given learned model $\hat{p}$ and true model p, we define perplexity $\ell(p, \hat{p})$ as the following.

$$\ell(p, \hat{p}) = 2^{H(p,\hat{p})},$$

where $H(p, \hat{p}) = \mathbb{E}_{w_{1:T} \sim p} [\log \hat{p}(w_1, ..., w_T)]$ is the cross entropy from p to $\hat{p}$. The expected value is estimated using a dev or test set.

n-Gram Models By setting $\lambda_1 = 1$ and $\lambda_2 = \lambda_3 = 0$, we get the singular unigram model (analogous settings index the bigram and trigram models). The perplexities are reported below.

	Unigram	Bigram	Trigram
Train Perplexity	976.54	84.39	8.06
Dev Perplexity	892.25	504.65	2501.40
Test Perplexity	896.50	501.154	2378.55

While the model with the most capacity (i.e. trigram) achieves the smallest training error, it overfits significantly and achieves extremely high dev and test errors. The bigram model performs well, balancing capacity and the ability to estimate it effectively.

Smoothed Models We also use the smoothed interpolation of the models, by choosing various settings of $(\lambda_1, \lambda_2, \lambda_3)$.

	(0.1, 0.3, 0.6)	(0.2, 0.4, 0.4)
Train Perplexity	11.12	10.48
Dev Perplexity	310.09	253.69
Test Perplexity	308.36	257.87

Even weighting between bigram and trigram probabilities seemed to balance the bias-variance tradeoff.

5 Discussion

To answer some of the hypothetical questions from A2, using half of the data would likely increase perplexity on previously unseen data, as we are less able to fit the true distribution and risk increasing generalization error via overfitting. As for increasing the number of unknown tokens, we can consider taking it to extremes and replacing nearly all of the tokens with `UNK`. Such a replacement would make the empirical training distribution extremely easy to fit, and would make the dev and test empirical distribution very similar to that of the training set. Thus, the estimated perplexity would likely decrease. However, doing so biases the learned distribution, making it difficult to use for tasks like generation.