# Assignment 1

COMP 7607: Natural Language Processing - The University of Hong Kong

Fall 2023

Zoie comes from Beijing and she is learning English. She finds it sometimes difficult to decide which preposition she should use in some cases. For example, should she say "they live in a small town" or "they live at a small town"? One day she visited HKU and came across a student who happens to study COMP7607 this semester.

"You must be an expert in language!", said Zoie.

"We learned language model recently!", said the student with excitement.

"That's nice! Could you please write a program for me that tells me what preposition I should use in a sentence", asked Zoie with her eyes shining like stars.

"Sure! Anything!", said the student.

And here comes the **assignment 1**.

**Reading Materials**: We recommend that you read J&M Ch.3 and M. Collins, Notes 1 before doing this assignment.

**Resources**: You can access all related resources at `https://github.com/qtli/COMP7607-Fall2023/blob/master/assignments/A1/`.

**Code**: We provide a Jupyter Notebook template. You can use it on your local environment or Google Colab (recommended) to finish it. The programming language is Python, building on the provided starter code. Some useful packages you may need: `nltk`, `numpy`.

**Grade**: We will evaluate your grade based on the code quality, output results/log, and discussion. We don't evaluate your submission by perplexity in Section 1, but by better prediction accuracy in Section 2 increases your score. There are some optional problems in this assignment, and solving them increases your grade. Please don't waste your time manually searching for hyperparameters. You are more than welcome to introduce a new hyperparameter optimization method to find more reasonable values.

**Submit**: You should submit two files: `UniversityNumber.ipynb` file and final prediction file `University-Number.test.out` of Section 2 to Moodle. All code and discussion should be included in your submitted `.ipynb` file. Make sure your code does not use your local files so that the results are reproducible. Before submitting, please run your notebook and keep all running logs so that we can check the results.

## 1 $N$-gram Language Model

A language model is a probability function p that assigns probabilities to word sequences such as $\vec{w} =$ (i, love, hong kong). Let's envision $p(\vec{w})$ as the probability that, if you were to tune in to the radio at any random moment, the following four words would be "i love Hong Kong." This probability can be evaluated even in the context of a longer sentence, such as “the students of COMP7607 course says, i love Hong Kong more than ever.” Often, we analyze $p(\vec{w})$ to compare it with alternative sequences and determine our preference.

Formally, in the context of language modeling, each element $W$ belonging to the event space $\mathcal{E}$ represents a potential value of the infinite sequence of words that would be emitted from the radio once it is turned on. The notation $p(\vec{w})$ serves as a shorthand for $p(\text{prefix}(W, |\vec{w}|) = \vec{w})$, where $|\vec{w}|$ represents the length of the sequence $\vec{w}$. Consequently, $p(i, love, hong, kong)$ encapsulates the cumulative probability of all infinite word sequences $W$ that commence with the phrase “i love hong kong...”.

Suppose $p(\vec{w}) = w_1 w_2 \dots w_n$ (a sequence of n words). A trigram language model defines

$$p(\vec{w}) \overset{\text{def}}{=} p(w_1) \cdot p(w_2 \mid w_1) \cdot p(w_3 \mid w_1\ w_2) \cdot p(w_4 \mid w_2\ w_3) \cdots p(w_n \mid w_{n-2}\ w_{n-1})$$

on the assumption that the sequence was generated in the order $w_1, w_2, w_3, \dots$ (“from left to right”) with each word chosen in a way dependent on the previous two words. (But the first word $w_1$ is not dependent on anything, since we turned on the radio at an arbitrary moment.)

**Q1**: Expand the above definition of $p(\vec{w})$ using naive estimates of the parameters, such as

$$p(w_4 \mid w_2, w_3) \overset{\text{def}}{=} \frac{C(w_2\ w_3\ w_4)}{C(w_2\ w_3)}$$

where $C(w_2 w_3 w_4)$ denotes the count of times the trigram $w_2 w_3 w_4$ was observed in a training corpus.

*Note: In practice, we refer to simple parameter estimates like these as "maximum-likelihood estimates" (MLE). The advantage of MLE is that it maximizes the probability, or minimizes perplexity, of the training data. However, they tend to perform poorly on test data*

**Q2**: One could also define a kind of reversed trigram language model $p_{reversed}$ that instead assumed the words were generated in reverse order (“from right to left”):

$$p_{reversed}(\vec{w}) \overset{\text{def}}{=} p(w_n) \cdot p(w_{n-1} \mid w_n) \cdot p(w_{n-2} \mid w_{n-1} w_n) \cdot p(w_{n-3} \mid w_{n-2} w_{n-1}) \quad (1)$$

$$\cdots p(w_2 \mid w_3 w_4) \cdot p(w_1 \mid w_2 w_3) \quad (2)$$

By manipulating the notation, show that the two models are identical (i.e., $p(\vec{w}) = p_{reversed}(\vec{w})$) for any $\vec{w}$ provided that both models use MLE parameters estimated from the same training data (see Q1 above).

## 2 *N*-gram Language Model Implementation

In this section, you will build your own language model. We provide a dataset with three files containing many whitespace-tokenized sentences at https://github.com/qtli/COMP7607-Fall2023/blob/master/assignments/A1/data/lm:

- train.txt: data for training your language model.
- dev.txt: data for developing and optimizing your language model.
- test.txt: data for only evaluating your language model.

You will first build vocabulary with the training data, and then build your own unigram, bigram, and trigram language models with some smoothing methods.

## 2.1 Building Vocabulary

You will download and preprocess the tokenized training data to build the vocabulary. To handle out-of-vocabulary (OOV) words, you will convert tokens that occur less than three times in the training data into a special unknown token `<UNK>`. You should also add start-of-sentence tokens `<s>` and end-of-sentence `</s>` tokens.

Please show the vocabulary size and discuss the number of parameters of $n$-gram models.

## 2.2 $N$-gram Language Modeling

After preparing your vocabulary, you are expected to build bigram and unigram language models and report their perplexity on the training set, and dev set. Please discuss your experimental results. If you encounter any problems, please analyze them and explain why.

## 2.3 Smoothing

In this section, you will implement two smoothing methods to improve your language models.

### 2.3.1 Add-one (Laplace) smoothing

Please improve your bigram language model with add-one smoothing. Report its perplexity on the training set and dev set. Briefly discuss the differences in results between this and the bi-gram model you built in Section 2.2.

### 2.3.2 Add-$k$ smoothing

One alternative to add-one smoothing is to move a bit less of the probability mass from the seen to the unseen events. Instead of adding 1 to each count, we add a fractional count k (.5? .05? .01?). This algorithm is therefore called add-$k$ smoothing, as shown here in the bigram case:

$$P^*_{\text{Add-k}}(w_i|w_{i-1}) = \frac{C(w_{i-1}\, w_i) + k}{C(w_i) + kV} \tag{3}$$

Please optimize the perplexity on the dev set by trying different $k$ (no more than three times). Report its perplexity on the training set and dev set. Briefly discuss the differences in results between this and the bi-gram model with add-one smoothing.

### 2.3.3 Linear Interpolation

Please implement linear interpolation smoothing between unigram, bigram, and trigram models. Report its perplexity on the training set and dev set. Please optimize on a dev set by trying different hyperparameter sets. Finally, report the perplexity on the test set with the best hyperparameter set you get. Briefly discuss the results.

**Optimization.** So far, we manually choose the hyperparameter that maximizes the perplexity of the dev set and evaluates it on the test set. There are various ways to find this optimal set of hyperparameters. Do you know any other learning algorithms? Please give an example.

## 3 Preposition Prediction

In this section, you will use your language model to answer what preposition to use in a sentence. We provide you with three files at `https://github.com/qtli/COMP7607-Fall2023/tree/master/assignments/A1/data/prep`:

- dev.in: The whitespace-tokenized sentences for developing your language model, where 5 prepositions (at, in, of, for on) are replaced with a special token <PREP>.
- dev.out: The correct answers for the dev set.
- test.in: Input sentences for testing in the same format as `dev.in`, which will be released a week before the deadline.

Your task is to use your language model to predict for each special token <PREP> in the validation (and soon the test) set, which preposition is the proper one there. You should optimize the accuracy of your model on the dev set and predict the prepositions of sentences in the test set. Please strictly follow the format of `dev.out` for your output `test.out` as well.

Please explain your method, report the accuracy of your model on dev sets, and discuss the results. Based on your hyperparameters and results, explain the relationship between the test perplexity model and performance in applications.

Go for it, Zoie. The journey of language modeling is about to start!