COS 484: Natural Language Processing **(Due: 02/17/26)**

Assignment #1

Instructor: Tri Dao, Karthik Narasimhan 120 points

Course Policy: Read all the instructions below carefully before you start working on the assignment and before you make a submission. The course assignment policy is available at `https://princeton-nlp.github.io/cos484/`. When you're ready to submit, please follow the instructions found here: COS 484 - Assignment Logistics.

- This assignment contains 2 parts, a theoretical and a programming part. The former consists of 3 problems, and the latter has 2, for a total of **5 problems**.
- We *highly* recommended that you typeset your submissions in LaTeX. Use the template provided on the website for your answers. If you have never used LaTeX, you can refer to the short guide here: Working with LaTeX. Include your name and NetIDs with your submission. If you wish to submit hand-written answers, you can scan and upload the pdf.
- Assignments must be uploaded to Gradescope by **11:59pm Eastern** on the due date mentioned above.
- As per the late-day policy outlined on the course website, you have 4 late days that you can use any time during the semester, with at most 3 late days per assignment. Once you run out of late days, late submissions will incur a penalty of 10% for each day, up to a maximum of 3 days beyond which submissions will not be accepted.
- All programming problems in this class should be completed in Google Colab using Python. If you would like to get familiar with this environment, you may complete the problems in this introductory Colab notebook (This will not be graded). If you've never worked with Google Colab before, take a look through this introduction guide: Working With Colab. **The answers to the written questions proposed in the programming part should be answered in your Colab notebook.**
- **LLM usage policy**: You **may not** consult a Large Language Model (LLM) when working on the **Theoretical** part of this assignment. For the **Programming** Part only, you may use coding assistants (like GitHub Copilot, Cursor, etc.) for writing code. If you do use such assistants, include the prompts you used at the end of your Colab notebook.

Theoretical Part

Submission Policy: Submit a single PDF for the answers to all questions in this part.

Problem 1: Language models and perplexity (10 + 5 + 5 + 10 = 30 points)

Assume you are given the following corpus of text:

```
<s> I like strawberries </s>
<s> You like all strawberries </s>
<s> I hate sour fruits </s>
<s> You like all sweet fruits </s>
<s> I like chocolate covered strawberries </s>
```

where `<s>` and `</s>` are tokens representing the start and end of a sentence, respectively. In all the following questions, **ignore** the probabilities $\mathbb{P}(\texttt{<s>})$ or $\mathbb{P}(\texttt{<s>} \mid \texttt{</s>})$.

(a) Provide the equation for bigram probabilities and estimate all **non-zero** bigram probabilities for this corpus (split into tokens by whitespace only, e.g `raspberries` and `berries` are different tokens). Present the probabilities in a tabular format:

Bigram	**Probability**
$\mathbb{P}(\texttt{I} \mid \texttt{<s>})$	.
.	.

(b) Using the above estimate, provide the probabilities for the sentences `<s> You like all strawberries </s>` and `<s> You hate sour fruits </s>`. Since the former is one of the five sentences in the corpus, would you expect its probability to be $\frac{1}{5}$? Why or why not?

(c) For any generic corpus, can you calculate accurate trigram probabilities for all word combinations $\mathbb{P}(w_3 \mid w_1, w_2)$ if you are given all bigram probabilities - $\mathbb{P}(w_3 \mid w_2)$ (without access to the true bigram/trigram counts)?

(d) Calculate the perplexity of the following corpus using your bigram model. Remember that `<s>` should not be included when calculating the length of the sentence.

```
<s> I like all sweet fruits </s>
<s> You like strawberries </s>
```

Problem 2: Naive Bayes (10 + 10 = 20 points)

Given the following documents with counts for key sentiment words, with positive or negative class labeled:

doc	"good"	"poor"	"great"	"terrible"	label
D_1	2	1	3	0	$+$
D_2	0	2	0	1	$-$
D_3	1	3	0	0	$-$
D_4	1	5	2	0	$-$
D_5	3	0	3	0	$+$
D_6	0	0	0	1	$-$

(a) Compute a naive Bayes model (with add-1 smoothing) on the above documents and assign a class ($+/-$) to the following sentence:

```
great characters and good acting, but terrible plot
```

You should just use $V = \{\text{good}, \text{poor}, \text{great}, \text{terrible}\}$. Provide details for your answer. *Note that this problem doesn't require programming.*

(b) For sentiment classification and a number of other text classification tasks, whether a word occurs or not matters more than its frequency. A variant of naive Bayes, called **binarized naive Bayes**, is to clip the word counts in each document at 1 (i.e. if a word appears multiple times in a document, it should be only counted once). Compute a binarized naive Bayes model (with add-1 smoothing) on the same documents and predict the label of the above sentence while providing details for your answer. Which of the two models do you think is better and why?

Problem 3: Understanding word2vec (5 + 5 = 10 points)

Given a sequence of words $w_1, \ldots, w_T$ and context size c, the training objective of skip-gram that we saw in class is:

$$\mathcal{L} = -\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\le j\le c, j\ne 0}\log P(w_{t+j} \mid w_t),$$

where $P(w_o \mid w_t)$ is defined as:

$$P(w_o \mid w_t) = \frac{\exp\left(\mathbf{u}_{w_t}^{\intercal}\mathbf{v}_{w_o}\right)}{\sum_{k\in V}\exp\left(\mathbf{u}_{w_t}^{\intercal}\mathbf{v}_k\right)},$$

where $\mathbf{u}_k$ represents the "target" vector and $\mathbf{v}_k$ represents the "context" vector, for every $k \in V$.

(a) Derive the following gradient (probability w.r.t context vector):

$$-\frac{\partial \log P(w_o \mid w_t)}{\partial \mathbf{v}_{w_o}}$$

(b) Assume we train this model on a large corpus (e.g. English Wikipedia). Describe at least two effects of choosing different context sizes c for training the word vectors $\mathbf{u}_w$, e.g. what would you expect if we used context size $c = 1$, 5, or 100?

Programming Part

Submission Policy: The answers to the written questions for this part should be answered in the Colab notebook. **Do not include the answers to this part in the same PDF as the theory part.**

Problem 1: Smoothing in language models

(5 + 5 + 5 + 10 + 5 = 30 points)

This part will require programming in Python 3. You will explore building and testing a language model with smoothing. To get started, open this Colab notebook.

You can download training and validation datasets for this problem from the links below:

- **Training data**: https://princeton-nlp.github.io/cos484/assignments/a1/brown-train.txt
- **Validation data**: https://princeton-nlp.github.io/cos484/assignments/a1/brown-val.txt

(a) Write two functions to tokenize the corpus. One function, `basicTokenize`, should simply split the text using whitespace. The other function, `nltkTokenize`, should use NLTK's word tokenization (`https://www.nltk.org`). Write another function to count the top k most frequent words in a list. Report the top 10 words ordered by their frequency in the training corpus, both using `basicTokenize` and `nltkTokenize`. What differences do you notice between the two?

(b) Using the `nltkTokenize` function you wrote, make a plot of the frequencies of words in the training corpus, ordered by their rank, i.e. the **most frequent** word first, the second most word next, and so on on the x axis. Plot only the top 100 most common words to see the trend more clearly. What pattern do you observe in your plot regarding frequency and rank? Do the frequencies follow Zipf's law? (Please fill code in the cell titled `"Code for sub-part (a)(b)"`).

Use the `basicTokenize` function and bigram language model ($n = 2$) for the following questions.

(c) Train the model and report its perplexity on the train and validation sets. Is the train or val perplexity higher and why? What do you notice about the val perplexity and why is this the case? (Please fill code in the cell titled `"Code for sub-part (c)"`).

(d) Implement Laplace (add-α) smoothing and retrain the model. Plot the perplexity on train and validation sets as a function of alpha (with values $10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 1, 10$). What happens to the validation and training perplexity as we increase alpha and why does this happen? What seems to be a good setting for alpha? Provide brief justification. (Please fill code in the cell titled `"Code for sub-part (d)"`).

(e) Based on your performance in the previous experiments, propose one idea apart from Laplace smoothing to improve the performance of your bigram language model on the validation set. Briefly describe the modification, explain why you expect it will improve validation perplexity, and discuss any potential limitations.

Problem 2: Sentiment analysis (15 + 10 + 5 = 30 points)

This problem will require programming in Python 3. The goal is to build a logistic regression model that you learnt from the class on a real-world sentiment classification dataset. To get started, open this Colab notebook.

The dataset you will be using is collected from movie reviews online. All the examples have been already tokenized and lowercased and each example is labeled as 1 (positive) or 0 (negative). The dataset has been split into a training, a development and a test set. You will only need the training and development sets for this problem, which you can download from the links below:

- **Training data**: https://princeton-nlp.github.io/cos484/assignments/a1/train.txt
- **Development data**: https://princeton-nlp.github.io/cos484/assignments/a1/dev.txt

Each line in the data files consists of a label (0 or 1) followed by the words in the sentence, separated by spaces. You will need to write a function to parse these files.

You will now build a Logistic Regression model for sentiment analysis. You are required to implement the full pipeline, including data loading, feature extraction, a logistic regression model, and the training loop.

- **Data Loading and Feature Extraction**: You will need to implement a method that processes raw text into feature vectors by mapping vocabulary terms to unique indices. Your implementation needs to support Unigram extraction, where features represent individual word counts, as well as Bigram extraction, where features represent consecutive word pair counts. Make sure your setup correctly handles feature indexing so that the same mapping is applied to both training and development data.
- **Model Implementation**: You should implement a class that supports the logistic regression logic. This includes:
  - **Initialization**: A function to initialize the model parameters (weights and biases) as well as hyper-parameters (including the learning rate, regularization parameter, and number of epochs).
  - **Optimization**: A training method that iterates through the dataset, calculates the gradient of the loss function for each example or batch, and updates the parameters using your chosen optimization function (we suggest using Stochastic Gradient Descent or Mini-batch SGD for efficiency).
  - **Inference**: A function that outputs the model's prediction for a single example.
- **Training Loop**: You should implement the logic for your model to train on the given training examples. Experiment with different hyperparameters to find the ones that optimize performance.

(a) In this part, we want to train the logistic regression model without regularization. Train your model with (i) unigram features only and (ii) bigram features only (two different models)

Report training and development accuracies for both runs on the dataset. How do the results of the unigram and bigram models compare? (Please fill code in the cell titled `"Code for sub-part (a)"`).

(b) Next, we will experiment with l_2 regularization $R(\theta) = \alpha\|\theta\|^2$. Fill in the code for performing logistic regression with regularization. Plot the accuracy on train and development sets as a function of $\alpha = \{0, 10^{-2}, 10^{-1}, 1, 10\}$. You only need to experiment with unigram features for this part. Explain what you observe. Does this match what you would expect from regularization? *Hint: There are several ways to implement the logistic regression model with regularization, and we will be accepting any correct implementation — regardless of the final model performance.* (Please fill code in the cell titled `"Code for sub-part (b)"`).

(c) Based on your model's performance in the previous experiment, propose one change you would consider making to either the model or feature extraction pipeline to further improve development set performance. Briefly describe the modification, explain why you expect it will improve validation perplexity, and discuss any potential limitations.