CSE556 NLP, Winter 2026
Assignment 1

Total Marks: 100 **Due:** 1 February 2026, 11:59 PM

Instructions

1. Please see the course policy on extension days and bonus points. There will be no deviation from the policy for any late submissions.
2. You may use LLMs, but you must provide the final prompt you used in your submission.
3. Wherever required, each training run should conclude within 15 minutes on your own device or in an online runtime.
4. Only one member of the group should submit the assignment as a ZIP file on Classroom.
5. All members of the group will be expected to be familiar with all parts of the assignment for the demo.
6. Doubts will be resolved only through the Google Form shared on Classroom.
7. All questions are of equal marks.

Q1 WordPiece Tokenizer Implement the WordPiece tokenizer components as follows:

1. The **preprocessor** should use NFKC normalization on the input text. After normalization, it must separate all punctuation symbols from surrounding characters with whitespace.
2. The **trainer** should accept the pre-processed corpus and a target vocabulary size. If multiple pairs have the same likelihood score during merging, break ties by selecting the pair that appears first in element-wise lexicographic order.
3. The **encoder** should use the learned vocabulary to tokenize input text using the greedy longest-match-first strategy. Use the prefix `##` to denote subwords that do not appear at the start of a word. The vocabulary should be ordered as follows:
   - Reserve the first 3 token IDs for special tokens: 0=`<UNK>`, 1=`<s>` (BOS), 2=`</s>` (EOS). BOS and EOS should be added around each sentence.
   - Continue assigning token IDs to the rest of the learned vocabulary in lexicographic order.
4. The **decoder** should reconstruct the source text from a sequence of tokens. It should merge subwords starting with `##` to the preceding token and join the remaining tokens with whitespace.

A subset of the WMT News Crawl containing Hindi text has been provided alongside this assignment. Your implementation must allow the following operations:

1. Accept an unprocessed input file of the same format, and output a text file containing the corresponding pre-processed lines.
2. Accept an unprocessed input file of the same format along with a target vocabulary size, and output a text file containing each token of the learned vocabulary on a separate line, ordered by token ID.
3. Accept an unprocessed input file of the same format along with a vocabulary file, and output a text file containing the space-separated tokens for each input line.
4. Accept an unprocessed input file of the same format along with a vocabulary file, and output a text file containing the space-separated token IDs for each input line.
5. Accept an input file containing space-separated token IDs along with a vocabulary file, and output a text file containing the corresponding detokenized (decoded) text.

Q2 FastText Word Representations FastText [1] represents each word as a bag of character n-grams, and its vector representation as the sum of the representations of each of the individual character n-grams. Implement the FastText word representation learning in two parts:

1. The **n-gram generator** should add word boundary symbols and convert words into a bag containing character n-grams of lengths 3 to 5, and a special sequence: the word with boundary symbols. Punctuation symbols should be separated by whitespace before processing the words. The n-grams should be processed as follows:
   - All distinct n-grams present in the input should be mapped to unique integer indices.
   - Assign indices to the n-grams and special sequences lexicographically in ascending order.
2. The **trainer** should implement the word2vec skipgram model with negative sampling and subsampling of frequent words (see [2]). Optionally, you may also use the negative sampling strategy and context window sampling specified in [1]. You may maintain the vectors as you wish.

Use the corpus provided in Q1. Your implementation must allow the following operations:

1. Accept an input file of the same format, and output a text file containing the space-separated bags for each input line. Each bag must contain the comma-separated IDs of the constituent n-grams of the corresponding word, sorted in increasing order of ID.
2. Save the word representations and perform the following experiments:
   i. Demonstrate how FastText allows representing words that would be OOV for a word2vec model without subword segmentation. (You may hardcode an example, and it need not be a real word).
   ii. Conduct the analysis in Section 6.2 in [1] and report whether the most important n-grams roughly correspond to morphemes.

Q3 Neural Probabilistic Language Model The Neural Probabilistic Language Model (NPLM) [3] approximates the conditional probability $P(w_t|w_{t-n+1}, \ldots, w_{t-1})$ using a feed-forward neural network operating on learned distributed word representations. Implement the NPLM architecture as follows:

1. The **input pipeline** should use the Q1 encoder to generate a sliding-window dataset with a configurable context size n. Reserve the last 10% of the corpus as a validation set. You must use the provided vocabulary file for this question. This vocabulary may differ from your Q1 output. Map any out-of-vocabulary words to the `<UNK>` token. Use the `<s>` token to pad the context window.

2. The **model** should implement the feed-forward architecture comprising a lookup table, concatenation of context vectors, a hidden layer with non-linear activation, and an output projection layer.

3. The **trainer** should train the model on the provided corpus. During training, replace context tokens with `<UNK>` with a probability of 0.01. It must save the learned model weights and any necessary metadata to disk for future inference.

4. The **inference** module should allow greedily generating text given a seed phrase.

Perform the following ablations and include the corresponding perplexity scores, next-token prediction accuracy values, and training/validation loss curves for each run in your report:

1. Compare the model performance across two different context sizes.

2. Compare two distinct model architectures.

Your implementation must allow the following operations:

1. Accept an input file of the same format as the training data, and output the perplexity and next-token prediction accuracy.

2. Accept an input file containing seed sentences (one per line) and a generation length k, and output a text file containing the original text continued with the generated text for each line.

Deliverables

Q1 Artifacts

- The complete source code used for the tokenizer, including the preprocessor, trainer, encoder, and decoder modules.
- The pre-processed corpus file generated by your normalization pipeline.
- A learned vocabulary file of size 10000, containing the subword tokens in the specified order.
- The tokenized text file containing space-separated tokens for the corpus.
- The token ID file corresponding to the tokenized text sample.
- The decoded text file demonstrating the reconstruction capability of the decoder.

Q2 Artifacts

- The source code for the FastText implementation, including the n-gram generator and the skip-gram trainer.
- The generated bags file containing the n-gram indices for the training corpus.
- The saved model weights and metadata files necessary to reload the model for inference.

Q3 Artifacts

- The source code implementing the NPLM architecture, training loop, and inference mechanism.
- The saved model checkpoints (weights) and associated metadata for both the context-ablation and architecture-ablation models.

Report

- A comprehensive explanation of the implementation steps, design decisions, and any assumptions made or references used for all three questions. Also explicitly state the work distribution among group members.
- For Q2: The training loss curves plotting loss against epochs.
- For Q2: The qualitative analysis of OOV word representation and morphological alignment as described in the instructions.
- For Q3: A detailed description of the model architectures and hyperparameters used.
- For Q3: The training and validation loss curves for all experimental runs.
- For Q3: A table reporting the final perplexity and next-token accuracy on both training and validation sets.

References

[1] Piotr Bojanowski et al. "Enriching Word Vectors with Subword Information". In: *Transactions of the Association for Computational Linguistics* 5 (2017). Ed. by Lillian Lee, Mark Johnson, and Kristina Toutanova, pp. 135–146. DOI: 10.1162/tacl_a_00051. URL: https://aclanthology.org/Q17-1010/.

[2] Tomas Mikolov et al. "Distributed Representations of Words and Phrases and their Compositionality". In: *Advances in Neural Information Processing Systems*. Ed. by C.J. Burges et al. Vol. 26. Curran Associates, Inc., 2013. URL: https://proceedings.neurips.cc/paper_files/paper/2013/file/9aa42b31882ec039965f3c4923ce901b-Paper.pdf.

[3] Yoshua Bengio, Réjean Ducharme, and Pascal Vincent. "A Neural Probabilistic Language Model". In: *Advances in Neural Information Processing Systems*. Ed. by T. Leen, T. Dietterich, and V. Tresp. Vol. 13. MIT Press, 2000. URL: https://proceedings.neurips.cc/paper_files/paper/2000/file/728f206c2a01bf572b5940d7d9a8fa4c-Paper.pdf.