CSE 556: Natural Langugae Processing Assignment 1

Deadline - 02 Feb 2025, Friday (11:59 pm IST)

Instructions

1. **Plagiarism** will be very strictly dealt with, and institute policy will apply.
2. Tasks are inter-dependent on each other, so please start timely. Each member is expected to contribute to a task individually and should be aware of all other tasks.
3. You have to submit a single .zip file named **A1-{Group No.}.zip**. Example: **A1-4.zip**.
4. **Code clarity and comments:** You will be also evaluated on how organized and well-commented your code is. (This component is here because tasks are inter-linked).
5. **Viva:** During the demo, some questions related to the concepts covered in this task and your implementation will be asked. The marks for the viva will not be based on group they will be assigned individually.
6. **End Note:** One does not simply dive into neural networks without experience. But worry not! Like the Fellowship of "The Ring", you'll need courage, time, and an early start to conquer this task. :)

Task 1 - Implement WordPiece Tokenizer — 30 Marks

In this task, you will implement **WordPiece Tokenizer FROM SCRATCH** using only standard Python libraries, as well as `NumPy` and `pandas` if needed. The use of any external NLP libraries or frameworks (such as NLTK, Spacy, TextBlob, HuggingFace, etc.) is strictly prohibited. Stick to the implementation taught in this [Tutorial].
You need to create a class named `WordPieceTokenizer` which should include the following methods:

1. `preprocess_data` - Implement this method to handle all the necessary preprocessing of the input data. Apply standard data processing techniques, but avoid using lemmatization or stemming, as they require external libraries.
2. `construct_vocabulary`: Using the provided text corpus, implement this method to create a vocabulary of tokens. Save the resulting vocabulary in a text file named `vocabulary_{Group_no.}.txt`, where each line contains a unique token.
3. `tokenize` - Implement this method to tokenize a given sentence into a list of tokens.

Deliverables

1. Submit a Python script named **task1.py** that contains the `WordPieceTokenizer` class and the implementation.
   - Your script should be capable of tokenizing samples from a provided `test.json` file, similar to "sample_test.json" (containg `id: sentence` pairs).

- During the demo, you code should generate a `tokenized_{Group_No.}.json` file, containing an ordered list of tokens in the format `{id: [token1, ....], ....}`

2. You need to submit the `vocabulary_{Group_no.}.txt` file, similar to "sample_vocabulary.txt", which lists all the unique tokens extracted from the courpus, with each tokens on a new line.

Task 2 - Implement Word2vec 25 Marks

You are tasked with building a pipeline for training a **Word2Vec** model using the CBOW (Continuous Bag of Words) approach **FROM SCRATCH** in PyTorch. It consist of the following components:

1. You are required to create a Python class named `Word2VecDataset` that will serve as a custom dataset for training the Word2Vec model. The implementaion should include the following components:

   - The custom implementation should work with PyTorch's DataLoader to efficiently load the training data.. You can refer this guide [Tutorial] on creating custom dataset classes in PyTroch.
   - `preprocess_data` - In this method, you will be preprocessing the provided corpus and prepare the CBOW training data for training the Word2Vec model.
   - During preprocessing, you must use the `WordPieceTokenizer` implemented in Task 1 to tokenize the input text corpus.

2. You required to create a Python class named `Word2VecModel` which implement Word2Vec CBOW architecture from scratch using PyTorch. After training the the model, save the trained **model's checkpoint** for later use.

3. Develop a function named `train` to manage the entire training process of the Word2Vec model. This function should include all the training logic.

Deliverables

1. Submit a Python script named **task2.py** that contains:

   - The implementation of both the `Word2VecDataset` and `Word2VecModel` classes.
   - The `train` function to handle the training process.
   - The code to run the training pipeline within the same script.

2. Provide the saved model checkpoint generated after training your `Word2Vec` model.

3. Plot graph showing training loss and validation loss versus epochs.

4. Identify two triplets as following:

   - Each triplet should include two similar tokens with high cosine similarity and one dissimilar token with low cosine similarity.
   - Compute cosine similarity using the word embeddings produced by `Word2Vec` model.

Task 3 - Train a Neural LM 45 Marks

For this task you need to train a **Neural Language Model** (Neural LM) which will be MLP based using *PyTorch*. The task consists of the following steps:

1. Implement a `NeuralLMDataset` that handles data preparation for training the langugae model. The implementation must include the following components:

   - The custom dataset must be compatible with PyTorch's `DataLoader`, similar to Task 2.
   - `preprocess_data` method to preprocess the text corpus for the next-word prediction task.

- Integration with the `WordPieceTokenizer` (from Task 1) and the `Word2VecModel` (from Task 2) to tokenize the corpus and create embeddings. You have to use the `Word2VecModel` checkpoint saved in Task 2)

2. Create three variations of the Neural LM architectures. The implementation must should be as follows:
   - Implement the classes named `NeuralLM1`, `NeuralLM2`, `NeuralLM3` for three different varations.
   - The three different architectures should differ logically, with changes such as different activation functions, increased number of layers, varying the number of input tokens etc.
   - For each variation, provide explanantions for the changes, their purpose, and the performance improvements gained. Ensure the changes are meaningful and not repetitive.
3. Develop a function named `train` to handle the trainging of all three Neural LM architectures. This function should include all the training logic.

Deliverables

1. Submit a Python script **task3.py** that contains:
   - The `NeuralLMDataset` class and three Neural LM architectures.
   - The `train` function for training all three models.
   - The code to run the training pipeline within the same script.
2. Plot the graph of training and validation loss versus epochs for all three models.
3. Accuracy and Perplexity Scores
   - Implement functions to compute accuracy and perplexity from scratch.
   - Report accuracy and perplexity scores for training and validation data.
4. You will be provided with a "test.txt" file (with one sentence per line, similar to "sample_test.txt"), implement a pipeline to predict the next three tokens for each sentence from vocabulary. Ensure that this pipeline is included in your script.

Report

You are required to submit a detailed report covering each task, you must include the following sections:

1. **Task 1** - Explain what pre-processing applied to the corpus. Describe how the vocabulary was built rom the corpus. Use examples generated from your implementation to explain.
2. **Task 2** - Add the train and validation loss v/s epoch graphs. Provide the triplets identified and explanation of the cosine similarities.
3. **Task 3** - Add the train and validation loss v/s epoch graphs. Justify the design choices for each of the three architectures and explain how each modification impacted the model's performance. Report accuracy and perplexity scores for both the training and validation datasets for each model. Discuss any differences in performances across the architectures.
4. **Individual Contribution** - Provide individual contributions of each group member.
5. **References** - Include a list of references used, such as research papers, tutorials and documentation.

NOTE: Marks for the report are included in the tasks, so ensure you provide detailed explanations for each section.