Assignment 3: Static Word Embeddings

Course: Introduction to NLP

Deadline: February 21, 2025 | 23:59

General Instructions

1. The assignment must be implemented in Python.
2. The assignment must be done in PyTorch (you are supposed to use standard functions only; **you may not use the gensim library for this assignment.**)
3. You are allowed to use AI tools or any internet resources as long as your implementation is in accordance with the assignment guidelines.
4. A single .zip file needs to be uploaded to the Moodle Course Portal.
5. Please start early since no extension to the announced deadline would be possible.

1 Concerning Word Vectors

Many NLP systems employ modern distributional semantic algorithms, known as word embedding algorithms, to generate meaningful numerical representations of words. These algorithms aim to create embeddings where words with similar meanings are represented closely in a mathematical space.
Word embeddings fall into two main categories: frequency-based and prediction-based. Frequency-based embeddings utilize various vectorization methods such as count vector, TF-IDF vector, and co-occurrence matrix with a fixed context window.
Prediction-based embeddings, exemplified by Word2vec, utilize models like Continuous Bag of Words (CBOW) and Skip-Gram (SG). However, the computational complexity of Word2vec's training algorithm can be high due to gradient calculations over the entire vocabulary. To tackle this challenge, variants such as Hierarchical Softmax output, Negative Sampling, and Subsampling of frequent words have been proposed.

This assignment involves implementing a frequency-based model using SVD, and prediction-based models using CBOW and Skip-Gram.

2 Training Word Vectors

For each of the algorithms, train embeddings using the Brown Corpus. This may be downloaded directly through NLTK. Store the learned embeddings as .pt files.

2.1 SVD (5 marks)

Implement a word embedding model and train word vectors by first building a Co-occurrence Matrix followed by the application of SVD.

2.2 The CBOW algorithm (15 marks)

Implement a word2vec model and train vectors using the Continuous Bag Of Words (CBOW) with Negative Sampling.

2.3 The Skip Gram algorithm (15 marks)

Implement a word2vec model and train word vectors using the Skip Gram model with Negative Sampling.

3 Word Similarity (5 marks)

WordSim-353 is a dataset used to evaluate semantic models by estimating similarity scores at a word level. Each row in the dataset provides a pair of words, the similarity score on a scale from 0-10, where 0 is the most dissimilar, and 10 is the most similar. You can download the dataset from Kaggle.
For each word pair in the dataset, compute the similarity of the corresponding embeddings through Cosine Similarity. Skip the examples where either of the two words have not occurred in the training corpus. Compute the Spearman's Rank Correlation between the cosine similarity and the score provided in the dataset.

4 Analysis (10 marks)

Compare and analyze the performance of SVD, CBOW and skip-gram on the WordSim task. Write a detailed report explaining the benefits and limiations of these techniques. You are also encouraged to plot graphs and tables to support your results.

5 Submission Format

Zip the following into one file and submit in the Moodle course portal. File nameshould be <roll_number>_<assignment3>.zip, eg: 2022xxxxxx_assignment3.zip

1. Source codes for word vectorization model
    - `svd.py`: Train the word embeddings using SVD method and save the word vectors.
    - `skipgram.py`: Train the word embeddings using Skip gram method (with negative sampling) and save the word vectors.
    - `cbow.py`: Train the word embeddings using CBOW method (with negative sampling) and save the word vectors.
2. Source code for word similarity

    ```
    wordsim.py <embedding_path>.pt
    ```

    Compute the cosine similarities using the provided embeddings and output the Spearman Rank Correlation.
3. Learned word embeddings
    - `svd.pt`: Saved word vectors for the entire vocabulary trained using SVD.
    - `skipgram.pt`: Saved word vectors for the entire vocabulary trained using Skip-gram (using negative sampling).
    - `cbow.pt`: Saved word vectors for the entire vocabulary trained using CBOW (using negative sampling).
4. The report containing your analysis, hyperparameters used and graphs showing the results
5. A csv file containing your results from the Word Similarity section.
6. README on how to execute the files and load the word embeddings.

6 Grading

Evaluation will be individual and will be based on your viva, report and submitted code review. You are expected to walk us through your code, explain your experiments, and report. You will primarily be graded based on your conceptual understanding.

- Word vectorization
    - SVD (5)
    - CBOW (15)
    - Skip-gram (15)
- Word similarity (5)
- Report and analysis (10)
- Viva during evaluation (50)

Resources

1. Efficient Estimation of Word Representations in Vector Space
2. Skip-gram with negative sampling
3. CBOW with negative sampling