Natural Language Processing

Dr. Leila Safari

University Of Zanjan, Department of Computer and Electrical Engineering

Homework 2-1: Adventures with Word Embeddings
Due Date: Sunday 1404-09-23, 11:59 pm
In person Delivery: Tuesday, 12:30 pm
(Upload at the provided link in LMS)

The aim of this assignment is to furnish you with a solid practical and theoretical understanding of the inner workings of word embeddings. In the first part of the assignment, you will explore the effects of the various hyperparameters in word embedding algorithms. In the second part, you will use word embeddings in a classification task.

General instructions

You should budget at least a day just for your full set of experiments to run successfully. This likely means several days' worth of runtime in the debugging phase. We strongly recommend that you start coding as soon as possible, but at minimum a week in advance of the deadline.

1 Parameter Search

The hyperparameters you will be exploring are:

- Architecture: (CBOW, Skip-gram)
- Dimension: (50, 100, 300)
- Window Size: (5, 10)
- Epochs: 5 (Train all models for exactly 5 epochs)
- *Note:* Keep negative sampling fixed at 5 to reduce search space.

This results in 12 unique settings (2*3*2).

Evaluation Implementation:

You will not be provided with evaluation scripts. Instead, you must implement the evaluation loop yourself using Gensim's built-in methods:

- Use evaluate_word_pairs() for the **WordSim353** task.
- Use evaluate_word_analogies() for the **BATS** tasks.
- **Note:** You are responsible for downloading the standard **WordSim353** and **BATS** datasets (available on typical NLP repositories like GitHub or Kaggle) and loading them correctly for these functions.

Writeup

Your submission for this part of the assignment will consist of

(1) A table containing the results of your parameter search;

(2) A written analysis of your results.

Each row of the table should contain numerical results for one choice of algorithm and parameter setting. The columns should include algorithm, context window, dimension, number of negative samples, correlation on WordSim353, accuracy for four BATS categories (you pick the ones you think are most interesting from among the 9 low-level categories, the 4 high-level categories, or the total score), and accuracy on the win353 paraphrase corpus. The table should look something like this:

Architecture	Win.	Dim.	#Neg.samples	WordSim	BATS 1	BATS 2	BATS 3	Win353-Para
Skip-gram	5	100	5	47.05	0.01	0.02	0.03	0.01
…	…	…	…	…	…	…	…	

Table 1: An example results table.

Your writeup should at least address the following prompts, but you are also encouraged to include other interesting observations or hypotheses you have made.

1. Does larger dimensionality always equate to better performance? In which categories and for which models? Why do you think this is?
2. Does better performance on one task mean better performance on the others? Provide a hypothesis as to why or why not.
3. Was performance roughly similar across all analogy categories? If different, how did it vary? Why do you think you observed this variation? Perform a brief error analysis and compare errors across the BATS categories you selected for your table.

Qualitative Analysis In addition to the numerical analysis, you must perform a qualitative inspection to understand what the model is actually learning:

- Pick **3 polysemous words** (words with multiple meanings), such as "bank", "apple", or "run".
- Find the **5 nearest neighbors** for these words in your **best performing model** vs. the **worst performing model** from Part 1.
- **Analyze the Window Size Effect:** Compare the neighbors. How did the context window size affect the semantic vs. syntactic similarity? (e.g., Does a small window like window=2 yield "running" for "run", while a larger window like window=10 yields "marathon"?).
- Report these findings in a table and briefly discuss them.

1.1 Training corpus

Instead of the classic Brown corpus, you will train your embedding models on the WikiText-2 dataset (or text8). This dataset allows for capturing richer semantic relationships due to its larger vocabulary and diversity.

You can download it via torchtext or directly from HuggingFace datasets, or use the gensim.downloader API (e.g., api.load('text8')).

Preprocessing: The raw data requires cleaning. You must implement a preprocessing pipeline that removes punctuation, handles numbers (e.g., replace with <NUM>), and lowers all cases.

1.2 Implementation details

We recommend that you train your word2vec models using the Gensim package. You are welcome to code up your own implementation (e.g. in PyTorch) if you like, but if you choose to do this, you are responsible for ensuring that your implementation is correct.

1.3 Bonus

For up to five points of extra credit, you may also do an additional evaluation and analysis. You can evaluate the above parameter settings **on GloVe (in this case we recommend using the implementation of GloVe available from its website);** Alternatively, you may implement a novel modification to word2vec or GloVe and evaluate it for up to five points of extra credit.

Homework 2-2: Downstream Task Evaluation

One of the most popular usages of word embeddings is in classification tasks. Instead of relying on existing repositories, you will build a classifier from scratch to categorize text topics.

The Dataset: We have provided a dataset named AG_News_Subset.csv (or you can download the "AG News" dataset from Kaggle/Torchtext). This dataset contains news articles classified into 4 categories: World, Sports, Business, and Sci/Tech.

Your Task: You need to compare three different feature extraction methods for a Logistic Regression (or SVM) classifier:

1. **Baseline:** TF-IDF Vectorization (Bag-of-Words approach).
2. **Your Best Model:** Use the best performing Word2Vec model you trained in Part 1. Represent each document by **averaging** the word vectors of its tokens.
3. **Pretrained GloVe:** Use glove-wiki-gigaword-100 (available via Gensim). Represent documents by averaging the vectors.

Requirements:

- For the embedding-based methods, you must handle "Out-of-Vocabulary" (OOV) words appropriately (explain your strategy in the report).
- Report **Accuracy, Precision, Recall, and Macro-F1 Score** for all three methods in a comparison table.
- **Analysis:** Why does TF-IDF perform better or worse than averaged word embeddings in this specific task? Does the "World" category get confused with "Business" more often in one model than another? (Show a Confusion Matrix)

Good Luck!

TA Team (Moslem Amini, Rana Naibi)