Assignment 3: Lexical Semantics and Word Vectors

Instructor: Luke Zettlemoyer *CSED503 - Sp 26*

Due at 11:59pm PT, May 15th, 2026.
40 points max, 20% towards the final grade

In this assignment, you will learn about pre-trained word embeddings and how to use them to solve NLP tasks, with a focus on lexical semantics and the geometry of embedding spaces.

You will submit both your **code** and **writeup** (as PDF) via Gradescope. Remember to **specify your collaborators** (including AI tools like ChatGPT) and **how they contribute** to the completion of your assignment at the beginning of your writeup. If you work on the assignment independently, please specify so, too. (Link to Homework File, please make your own copy!!)

Required Deliverables

- **Code Notebook**: This assignment has one associated Jupyter notebook. Submit the notebook with your solutions as a Jupyter Notebook file (`.ipynb`) in Gradescope. On Google Colab you can do so by `File` → `Download` → `Download .ipynb`. **Please comment out any additional code you had written to solve the write-up exercises before submitting on gradescope to avoid timeouts.**
- **Write-up**: For written answers and open-ended reports, produce a single PDF for §1 and submit it in Gradescope. We recommend using Overleaf to typeset your answers in LaTeX, but other legible typed formats are acceptable. We do not accept hand-written solutions because grading hand-written reports is incredibly challenging.

Recommended Reading

The homework is based on chapter 6 and 7 of Jurafsky and Martin. You can also check the GloVe paper (Pennington et al. 2014), as we will be making heavy use of GloVe word vectors in this homework. For evaluating biases in word embeddings, you will be implementing the Word Embedding Association Test (WEAT), and we recommend reading Caliskan et al. 2017, to understand the method in detail.

Required Compute

You can run the notebook on Google Colab with a CPU runtime. If your implementation is optimized, all test cases should run fairly quickly. You can use a GPU runtime for faster runtimes, but please do not hardcode the device in PyTorch as `"cuda"` anywhere, as the Gradescope autograder may not use a GPU. Instead, you can dynamically set the device as `"cuda"` if the system where the code is being run has a CUDA-enabled GPU by running the following command:

```
DEVICE = "cuda" if torch.cuda.is_available() else "cpu"
```

Acknowledgement

This assignment is adapted by Daniel Kim based on work by Kabir Ahuja. Kavel Rao also helped design autograder for the homework.

1 Lexical Semantics (40 pts)

In this project, you will work with vector representations of words, explore intriguing and sometimes undesirable properties of the geometry of these learned representations. There are different variants for pre-trained word embeddings available, like word2vec, GloVe, and fastText. For this homework we will be working with GloVe vectors. GloVe is conceptually very similar to word2vec i.e. it learns vector representation of words such that semantically similar words are closer in the embedding space. The two mainly differ on the choice of objective to train the word vectors. We will later move on to building sentence level representations using word embeddings and use those to build a nearest neighbor text classifier.

Deliverables:

1. **Coding Exercises:** You should complete the code blocks denoted by `YOUR CODE HERE:` in the Python notebook. Do not forget to remove `raise NotImplementedError()` from the code blocks. To submit your code, download your notebook as an IPython notebook file (`CSED503_Assignment3.ipynb`).
2. **Write-up:** Your report for §1 should be **no more than four pages**. However, you will most likely be able to answer all questions within three pages.

1.1 Geometry of Word Embeddings

1.1.1 Background

Linear Representation Hypothesis. While it makes sense for the cosine similarity between word vector representations to correspond to semantic similarity as the way these vectors were trained were to capture exactly this. However, what is really surprising is that linear directions in the embedding space of the word vectors often corresponds to meaningful concepts (syntactic or semantic), as first observed in Mikolov et al. 2013. For e.g. , what they found was that distance between embeddings for king and queen is almost same as the distance between embeddings for man and woman. The vector difference between embeddings of king and queen i.e. $v(\text{king}) - v(\text{queen})$ can be thought as a direction representing the male to female gender direction, i.e. $v(\text{woman}) \approx v(\text{king}) - v(\text{queen}) + v(\text{man})$. They similarly, found other concept directions like singular to plural, $v(\text{queens}) \approx v(\text{kings}) - v(\text{king}) + v(\text{queen})$. This idea that higher level concepts are represented linearly as directions in the embedding space is called Linear Representation Hypothesis. This property is puzzling because the models were not trained to achieve such behavior. For readers curious about understanding an explanation of this phenomenon, we recommend checking the ICML paper by Allen and Hospedales 2019.

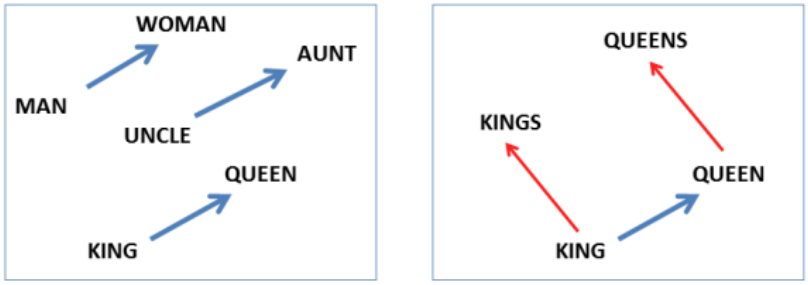

Figure 1: Example from Mikolov et al. 2013 demonstrating linear representation hypothesis.

Bias in Word Embeddings and WEAT. Word embeddings trained on large text corpora while capture useful semantic relationships between words, unfortunately also learn biases (e.g. societal) present in their training data. Caliskan et al. 2017 found human like semantic biases in trained word vectors like GloVE, e.g. it was found that African American names were found to be significantly closer (in the embedding space) to unpleasant terms (e.g. sickness, accident, abuse) than European American names which were closer to pleasant terms (e.g. love, peace, health). They propose the Word Embedding Association Test (WEAT) to measure such biases. The test is based on the Implicit Association Test (IAT) from psychology. It considers two sets of target words X and Y (e.g. flower names: 'marigold', 'poppy', 'azalea' and insect names: 'ant', 'caterpillar', 'flea') and two sets of attribute words A and B (e.g. pleasant terms: 'love', 'peace', 'health', and unpleasant terms: 'sickness', 'accident', 'abuse'). It then measures how separated are the two distributions of associations between the target and attribute (i.e. X's association with A and B, and Y's association with A and B). More formally, it defines a metric called *Effect Size*, which is given by:

$$\text{effect-size} = \frac{\text{mean}_{x \in X} s(x, A, B) - \text{mean}_{y \in Y} s(y, A, B)}{\text{std-dev}_{w \in X \cup Y} s(w, A, B)}$$

where $s(w, A, B)$ is given as:

$$s(w, A, B) = \text{mean}_{a \in A} \cos(\vec{w}, \vec{a}) - \text{mean}_{b \in B}(\vec{w}, \vec{b})$$

where $\cos(\cdot, \cdot)$ computes the cosine similarity between two vectors and $\vec{w}$ corresponds to the word vector for the word w.

Ideally we would want the effect size to be as close to 0 as possible, i.e., words in both target groups are equally close to the two attribute sets of words. A higher effect size indicates higher bias.

Whenever making claims about trends in data, it is very important to consider the statistical significance of the observations. WEAT also includes a permutation test to check significance of the null hypothesis: the words in set Y are closer to words in A than B, compared to X. This is done by first calculating the test statistic:

$$s(X, Y, A, B) = \sum_{x \in X} s(x, A, B) - \sum_{y \in Y} s(y, A, B) \tag{1}$$

This is called the differential association of words in target groups X and Y (towards the attribute sets). Intuitively, considering the flower-insect pleasant-unpleasant example, it measures the extent to which flower names are close to pleasant terms than unpleasant terms compared to insect names' association between the two groups.

It then considers different permutations of target groups X and Y by mixing the words between two groups, X_i and Y_i (hence, X_i not only just contains flower words but also some insect words, similarly Y_i also containing both). It then checks how often do the differential association of these permuted target groups is higher than the differential association of the original groups. If it happens a lot, then probably our results are not statistically significant, but if it happens very rarely that indicates the presence of the bias strongly. More formally, the *p-value* of this permutation test is given by:

$$\text{p-value} = \Pr[s(X_i, Y_i, A, B) > s(X, Y, A, B)]$$

We reject the null hypothesis if the p-value is very low (0.05 is a commonly used threshold), i.e., our results are statistically significant.

Don't worry if you do not understand all the math behind the permutation test. Just make sure that you at least have a basic intuition of how it works. Feel free to come for office hours if you need help in understanding this. We will provide you with the code to compute the p-value, you will only need to implement the functions for $s(w, A, B)$, effect-size, and $s(X, Y, A, B)$.

1.1.2 Coding Exercises

Implement the following classes and functions in the notebook:

- functions `cosine_similarity`, `euclidean_distance`, `manhattan_distance`, and `find_synonym` (5 pts)
- function `find_analogy_word` (3 pts)
- functions `word_association_wth_attribute` `weat_effect_size`, and `target_words_diff_association_wth_attribute` (10 pts)

1.1.3 Write-Up Questions

1. **Why cosine similarity works better than other distance metrics? (1 pt)** Can you provide an intuition about why cosine similarity did much better at capturing the semantic similarity between the words compared to Euclidean or Manhattan Distance?
2. **Does similarity in embedding space imply semantic similarity? (1 pt)** While finding synonyms using cosine similarity works well, you should see that it gets the last synonym question wrong (the answer should be positive):

   30. What is a synonym for sanguine?

   a) pessimistic

   b) unsure

   c) sad

   d) positive

   What word does it choose instead? In 1-2 sentences, explain why you think it got the question wrong.
3. **Finding Concepts in Embedding Space. (2 pts)** Play around with your `find_analogy_word` function and find instances of concepts other than the ones we tested your code already. You should provide at least 3 new concepts and 2 examples for each concept i.e. values of a, b, aa, bb and `choices_list` where your function selects bb from `choices_list`.
4. **Effect of Scale on WEAT Effect Sizes. (4 pts)** In the notebook, you used GloVE vectors of size 50 which were trained on 6B tokens of data from Wikipedia. How do you expect the effect sizes to change as we vary the size of vectors as well as amount of data? We provide you with glove vectors trained on 6B tokens of sizes 50, 100, 200, and 300. Similarly, we also provide you 300 dimensional glove vectors trained on 6B, 42B, and 840B tokens. These files can be found in the path (accompanying the code) `data/embeddings/glove.{{num_tokens}}/glove.WEAT.{{num_tokens}}.{{vector_size}}d.txt`. Using these, plot the effect sizes while keeping number of tokens fixed to 6B and varying the vector sizes, as well as fixing vector size (300) and varying number of tokens. Plot the effect sizes for the categories: `'EuropeanAmerican_AfricanAmerican_Pleasant_Unpleasant'`, `'Flowers_Insects_Pleasant_Unpleasant'`, `'Male_Female_Career_Family'`, `'Math_Arts_Male_Female'`, `'MusicalInstruments_Weapons_Pleasant_Unpleasant'`. Explain the trends you see in 2-3 lines.

1.2 From Word Embeddings to Sentence Level Embeddings

Until now we have been dealing with word level vector representations. However, for most of the interesting NLP problems what we are really interested in are sentence level or even document level representations. While learning sentence level embeddings requires more advanced neural architectures like Transformers, which we will soon see in the lectures, we can also use simple methods to arrive at sentence level representations that can often be strong baselines to compare against more complex methods. One of the simplest

ways to get a sentence embedding from word embeddings is to just sum word representations for all words appearing in the sentence, i.e. for a sentence s, with words $w_1, \cdots, w_n$, we have

$$\vec{s} = \vec{w_1} + \cdots + \vec{w_n}$$

We can alternatively take a weighted sum of the word vectors, where these weights can come from heuristics such as TF-IDF weights (which suppress the weights for very common words) or as you will implement in the coding exercise assign different weights based on the *Part of speech* tag of the word.

While this works reasonably well in practice for some use cases, it is clearly very limited, as it doesn't really account for the structure of the sentence, and is essentially like a bag of words representation like we saw in previous homework. We will see how we can used transformer based pre-trained sentence level embeddings using the `sentence-transformer` library, which are much more powerful as they learn contextual representations of the words in a sentence / document. This is mainly to demonstrate the importance of richer sentence level representations to you, you can treat these as black boxes for now, we will cover details about these models in coming weeks.

1.2.1 Coding Exercises

Implement your code for following classes and functions in the notebook.

- functions `get_sentence_embedding`, `get_sentence_similarity` (3 pts)
- class `GloveKNNClassifier` (4 pts)
- class `SentenceTransformerKNNClassifier` (3 pts)

1.2.2 Write-Up Questions

1. **Issues With Summed Word Embeddings as Sentence Representations. (1 pt)** You must have noticed that the nearest neighbor classifier with glove embeddings doesn't perform very good, in fact it performs worse than the the linear classifiers we trained in HW1. One of the reasons why this happens is because the way we construct the sentence embeddings by summing the word embeddings. Can you think of the issues with such an approach? Keeping the task of sentiment analysis in mind provide your reasoning in not more than 3-4 lines, with examples.

2. **How the number of nearest neighbors k effect KNN Classifier's performance? (3 pts)** Plot training and dev set accuracies (for both binary and multi class case) for different values of k (e.g $k = [1, 3, 5, 7, 9, 19, 29, 39, 49]$) for the K-nearest classifier trained using Glove Embeddings. What trends do you notice, can you explain why we see such trends? You can only show the plots for summing vectors approach here and need not provide plots for POS weighted representations. To get train and dev accuracies for a given value of k, you can run `accs = exercise5.evaluate(k = k, return_vals=True)`