Assignment 3: Lexical Sementics and Word Vectors

Instructor: Noah Smith *CSE P517 – Spring 2025*

Due at 11:59pm PT, May 12, 2025.
40 points

In this assignment, you will learn about pretrained word embeddings and how to use them to solve NLP tasks.

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.

Required Deliverables

- **Code Notebook**: The assignment is associated with Jupyter notebook. (CSE447_Assignment3.ipynb) Please download the notebook as Jupyter notebook files (`.ipynb`) and submit them 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 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 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. We provide all the details necessary to solve the homework in this handout and the notebooks, so it is not required to read the these to solve the exercises. However, we recommend going through them if you are confused about any concepts that are covered in the homework. You are also welcome to come to office hours if you have trouble understanding any of these concepts.

Required Compute

You can run the notebooks on Google Colab with CPU runtime. If your implementation is optimized, except a few cases, all test cases should run fairly quickly. You can use a GPU runtime for faster runtimes. Please do not hardcode the device on pytorch as `"cuda"` anywhere, as gradescope autograder won't be using 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 designed by Kabir Ahuja with invaluable feedback from Khushi Khandelwal, Melissa Mitchell, Kavel Rao, and Riva Gore. Kavel Rao also helped design autograder for the homework.

1 Lexical Semantics (40 pts)

In this assignment, 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 and GloVe. 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 tend to be 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:** The assignment is associated with Jupyter notebook.

   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 a Jupyter notebook file (CSE447_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 these vectors were trained to capture exactly this. However, what is really surprising is that linear directions in the embedding space of the word vectors often correspond to meaningful concepts (syntactic or semantic), as first observed by Mikolov et al. (2013). For example, what they found was that difference between embeddings for 'king' and 'queen' is almost same as the difference between embeddings for 'man' and 'woman'. The vector difference between embeddings of 'king' and 'queen', $v(\text{king}) - v(\text{queen})$, 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 also 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 the 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 Allen and Hospedales (2019).

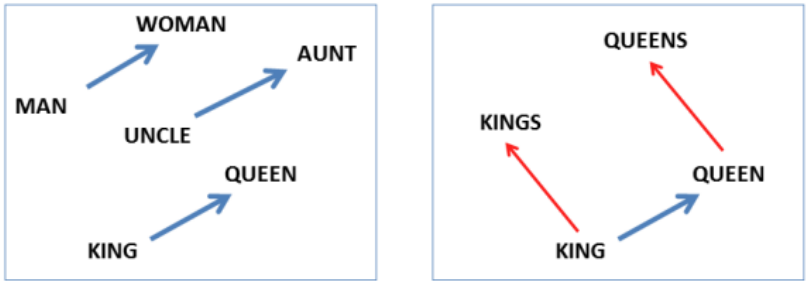

Figure 1: Example from Mikolov et al. (2013) illustrating the linear representation hypothesis.

Bias in Word Embeddings and WEAT. Word embeddings trained on large text corpora to capture useful semantic relationships between words, also learn societal biases present in their training data. Caliskan et al. (2017) found human-like semantic biases in trained word vectors like GloVe, e.g., 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. Applied to word vectors, 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:

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

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

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

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

Unbiased representations would have an effect size close to zero, 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 the 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 closer to pleasant terms than unpleasant terms, compared to insect names' association between the two groups.

The significance test 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). The method then checks how often 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 is evidence for bias. More formally, the p-value of this permutation test is an estimate of the probability:

$$\Pr(s(X_i, Y_i, A, B) > s(X, Y, A, B))$$

We reject the null hypothesis if the p-value is very low (0.05 is a commonly used threshold), i.e., if the difference is 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

Check section 1 of CSEP517_Assignment3_Writeup.pdf for detail.

1.2 From Word Embeddings to Sentence Level Embeddings

Until now we have been dealing with word-level vector representations. However, most NLP systems must deal with text formed into sentences or longer passages like documents. While learning the most effective sentence level embeddings requires more advanced neural architectures, we can also use simple methods to arrive at sentence level representations. (These are 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 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 tfidf 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 approach 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. 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 them as black boxes for now.

In this part of the assignment, you will use a classic type of classifer called k-nearest neighbors (KNN). Given a training set of labeled instances and a similarity function, with each one represented as a vector, this classifier takes a new input x, finds the k training instances that are most similar to x (using the similarity function), and returns the most common label in that set of k "neighbors." The performance of this kind of classifier depends heavily on how you represent the data (what we call embeddings in NLP), the similarity function, and the hyperparameter k.

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.

Check section 2 of CSEP517_Assignment3_Writeup.pdf for detail.