# Assignment 3

CSE 447 and 517: Natural Language Processing – University of Washington

Winter 2021 – Due: February 5, 2021, 11:59 pm

**Submit:** Your submission will be through Gradescope and will have two parts:

- Writeup for all problems *except* problem 3. Your writeup for the non-implementation problems will be turned in separately as a pdf with one-inch margins and 11-point Times font. Part of the training we aim to give you in this class includes practice with technical writing. Organize your report as neatly as possible, and articulate your thoughts as clearly as possible. We prefer quality over quantity.
- Problem 3. In a zip file (this is the only acceptable format!), you will submit your code, a neatly written README file to instruct how to run your code with different settings, and a pdf report containing your answers this problem's questions only. We assume that you always follow good practice of coding (commenting, structuring), and these factors are not central to your grade. You may implement the models in the programming language of your choice. However, please provide well commented code if you want partial credit. If you have multiple files, please provide a short description in the preamble of each file. Do not flood the report with tangential information such as low-level documentation of your code that belongs in code comments or the README. Similarly, when discussing the experimental results, do not copy and paste the entire system output directly to the report. Instead, create tables and figures to organize the experimental results.

## 1 CSE 447 and CSE 517 Students: Based on Eisenstein 14.4 (pp. 331–2)

A simple way to compute a vector representation of a sequence of words is to add up the vector representations of the words in the sequence. Consider a sentiment analysis model in which the predicted sentiment is given by:

$$\text{score}(w_1, \ldots, w_m) = \boldsymbol{\theta} \cdot \sum_{i=1}^{m} \mathbf{x}_{w_i} \tag{1}$$

where $w_i$ is the $i$th word and $\mathbf{x}_{w_i}$ is the embedding for the $i$th word; the input is of length $m$ (in word tokens). $\boldsymbol{\theta}$ are parameters. Prove that, in such a model, the following two inequalities cannot both hold:

$$\text{score}(\text{good}) > \text{score}(\text{not good}) \tag{2}$$

$$\text{score}(\text{bad}) < \text{score}(\text{not bad}) \tag{3}$$

Next, consider a slightly different model:

$$\text{score}(w_1, \ldots, w_m) = \frac{1}{m} \left( \boldsymbol{\theta} \cdot \sum_{i=1}^{m} \mathbf{x}_{w_i} \right) \tag{4}$$

Construct an example of a pair of inequalities similar to (2–3) that cannot both hold.

## 2 CSE 447 and CSE 517 Students: Based on Eisenstein 14.5 (p. 332)

Continuing in the style of problem 1, consider this model:

$$\text{score}(w_1, \ldots, w_m) = \boldsymbol{\theta} \cdot \text{ReLU}\left(\sum_{i=1}^{m} \mathbf{x}_{w_i}\right) \tag{5}$$

Show that, in this case, it *is* possible to achieve the inequalities (2–3). The recommended way to do this is to provide weights $\boldsymbol{\theta}$ and embeddings for the words *good*, *bad*, and *not*. The problem can be solved with four dimensions.
Hint: problems 1 and 2 are related to the "XOR" problem (2) from assignment 2.

## 3 CSE 447 and CSE 517 Students: Implementation Problem Based on Eisenstein 14.6–9 (p. 332)

1. Download a set of pretrained English word embeddings (e.g., the GloVe embeddings available at https://nlp.stanford.edu/projects/glove/). Use cosine similarity to find the most similar word to each of these words. Report the most similar word and its cosine similarity.

   - *dog*
   - *whale*
   - *before*
   - *however*
   - *fabricate*

2. Use vector addition and subtraction to compute target vectors for the analogies below. (This style of evaluation is explained in section 14.6 of the textbook. The textbook is, however, incorrect in the in-line formula that explains how to compute the target vector on page 322. In the textbook's notation, you want the word embedding most similar to $(-\mathbf{v}_{i_1} + \mathbf{v}_{j_1} + \mathbf{v}_{i_2})$. Note the sign changes. The same mistake appears in the video recording but has been corrected in the pdf version of the lecture slides.) After computing each target vector, find the top three candidates by cosine similarity. Report the candidates and their similarities to the target vector.

   - *dog* : *puppy* :: *cat* : ?
   - *speak* : *speaker* :: *sing* : ?
   - *France* : *French* :: *England* : ?
   - *France* : *wine* :: *England* : ?

3. Select a text classification dataset that is available to download for research (e.g., the Pang and Lee movie review data, currently available from http://www.cs.cornell.edu/people/pabo/movie-review-data/). If there is an official test dataset, set it aside; if not, use a random subset as the test set (at least 300 examples) and do not use it until part 4. Use the official development set as development data, if it exists; if not, use a random subset and make sure your training set and development set do not overlap. Train a classifier on your training set, conforming to these requirements:

   (a) Use the pretrained word embeddings from part 1 as inputs; *do not* update them during training (they are "frozen").

(b) For every word type in the training data that has no pretrained embedding, introduce a new word embedding, which is randomly initialized and updated during training. When you apply the model to development or test data, words that have no embedding (neither pretrained, nor added during training) should be ignored.

(c) Train until accuracy on the development set stops improving.

(d) You may use the development set to tune the model architecture (e.g., choosing a depth) and hyperparameters.

(e) Which kind of network you use is up to you. In the 447 discussion section, a GRU-based RNN was considered, and you're welcome to use that, or to explore other options if you're curious. You should briefly describe your network in the writeup.

4. Report accuracy, macro-averaged $F_1$ score (i.e., first, for each class, calculate $F_1$ with that class as target; then average those per-class $F_1$ scores), and training time. (You may also optionally report the training time summed across all the models you considered during architecture/hyperparameter tuning; note that doing this requires more planning.)

5. Repeat the above experiment—keeping everything the same (including tuning), except for one key change: update ("finetune") all word embeddings during training. Report the same measurements as in part 4.

## 4 CSE 517 Students: Problem Based on Eisenstein 14.2 (p. 331)

In skipgram word embeddings, where we have word vocabulary $\mathcal{V}$ and contexts $\mathcal{C}$, the negative sampling objective can be written as:

$$L = \sum_{i \in \mathcal{V}} \sum_{j \in \mathcal{C}} \text{count}(i, j) \psi(i, j) \tag{6}$$

$$\psi(i, j) = \log \sigma(\mathbf{u}_i \cdot \mathbf{v}_j) + \sum_{i \in \mathcal{W}_{\text{neg}}} \log(1 - \sigma(\mathbf{u}_{i'} \cdot \mathbf{v}_j)) \tag{7}$$

Equation 7 corresponds to equation 14.19 in the textbook, discussed in section 14.5. Please see the textbook for a full explanation of the notation.
Assume we draw the negative samples from the empirical unigram distribution (i.e., the relative frequencies of vocabulary words as observed in the training corpus).

1. Compute the expectation of $L$ with respect to the negative samples.

2. Take the derivative of this expectation with respect to the score of a single word context pair $\sigma(\mathbf{u}_i \cdot \mathbf{v}_j)$, and solve for the pointwise mutual information of word $i$ with context $j$.. You should be able to show that at the optimum, the PMI is a simple function of $\sigma(\mathbf{u}_i \cdot \mathbf{v}_j)$ and the number of negative samples.

This exercise is part of a proof that shows that skipgram with negative sampling is closely related to PMI-weighted matrix factorization; see Levy and Goldberg [2] along with Eisenstein [1] chapter 14 for more background.

## References


[1] Jacob Eisenstein. *Introduction to Natural Language Processing*. MIT Press, 2019.

[2] Omer Levy and Yoav Goldberg. Neural word embedding as implicit matrix factorization. In *NeurIPS*, 2014.