CSC 396: Assignment #4 (150 points)

Due by 11:59PM, November 23rd

Upload your assignment as a *new* repository in GitHub Classroom and send us the link

Note that this assignment has both a programming part (problems 1 and 2) and a theory part (problem 3). Please answer the questions in problem 3 in a separate document (Word, PDF, etc.) that is checked in your GitHub repository. The documentation for problems 1 and 2 can be included in the same document or as comments in the Jupyter notebook(s). Lastly, please include in the document an estimate of the number of hours you worked on each problem. You won't be graded on this estimate; I need it for course statistics.

Setup (10 points)

This assignment will be implemented from scratch in your own GitHub repository. Once completed, please email us the repository URL.

Problem 1 (60 points)

In this assignment, you will use a transformer encoder to implement static embeddings (like word2vec). The method we will use comes from this paper: https://aclanthology.org/2020.acl-main.431.pdf

Please implement the algorithm below:

1. Choose the transformer encoder you will work with. For example, these are good encoder models:
    a. https://huggingface.co/FacebookAI/roberta-base
    b. https://huggingface.co/microsoft/deberta-v3-base
    c. https://huggingface.co/Tejas3/distillbert_base_uncased_80_equal
2. Read the texts provided with this assignment in the dataset uploaded in D2L under Content / Assignments / assignment4-dataset.txt.gz. This dataset contains one sentence per line. Tokenize each individual sentence using the tokenizer corresponding to the transformer chosen in the previous step. Note that the resulting tokens are *sub-word* tokens that may not correspond to a full word.
3. Generate the contextualized embeddings for all the tokens in the dataset and compute the average embedding for each token in the vocabulary by averaging all its contextualized embeddings.

Problem 2 (50 points)

Implement the most_similar() function from the chapter 9 code, and use it to run the six examples in the notebook. Include the output of these calls in your notebook.

IMPORTANT NOTE: the most_similar() function operates over actual words, whereas the embeddings you computed in problem 1 operate over transformer tokens. That is, each English word may consist of one or more tokens. To aggregate token embeddings into word embeddings, implement the following algorithm:

1. Take the glove_vocabulary.txt (available in D2L under Content / Assignments) file and tokenize all the words in this file using the *same* tokenizer you used in the previous problem.
2. Compute a word embedding for all words in this file by averaging the corresponding token embeddings.

Problem 3 (30 points)

Consider the simplified self-attention algorithm below, where the only difference from the original algorithm is that it uses a simpler formula to normalize the attention weights a_{ij} in step (b) (instead of the original softmax). Formally, given the query, key, and value vectors for all words in the input text (i.e., $\mathbf{q}_i$, $\mathbf{k}_i$, $\mathbf{v}_i$ for word w_i), the self attention algorithm operates as follows:

(a) For each pair of words, w_i and w_j, compute the attention weight a_{ij} using the $\mathbf{q}_i$ and $\mathbf{k}_j$ vectors. In particular:

  i. Initialize the attention weights a_{ij} with the product of the corresponding query and key vectors: $a_{ij} = \mathbf{q}_i \cdot \mathbf{k}_j$.

  ii. Divide the above values by the square root of the length of the key vector: $a_{ij} = a_{ij}/\sqrt{|\mathbf{k}_1|}$ (all $\mathbf{k}_i$ vectors have the same size, so we arbitrarily use $\mathbf{k}_1$ here).

(b) For each word w_i, normalize a_{ij} by dividing it by the sum of the elements in $\mathbf{a}_i$: $a_{ij} = \frac{a_{ij}}{\sum_k a_{ik}}$.

(c) For each word w_i, multiply the above attention weights with the corresponding value vectors, for all words w_j. Then sum up all these weighted vectors to produce the output vector for w_i: $\mathbf{z}_i = \sum_j a_{ij}\, \mathbf{v}_j$.

Using the algorithm above, compute the $\mathbf{z}_1$ embedding for the token *bagel* in the following text with three tokens: *bagel with cheese*. Assume no other tokenization takes place. Use the following values for the $\mathbf{q}_i$, $\mathbf{k}_i$, $\mathbf{v}_i$ vectors (indexes start at 1):

$$\begin{array}{l|l|l} \mathbf{q}_1 = [1,2,3] & \mathbf{k}_1 = [1,1,1] & \mathbf{v}_1 = [2,0,1] \\ \mathbf{q}_2 = [2,3,2] & \mathbf{k}_2 = [0,0,0] & \mathbf{v}_2 = [3,0,0] \\ \mathbf{q}_3 = [5,6,7] & \mathbf{k}_3 = [2,2,0] & \mathbf{v}_3 = [1,2,2] \end{array}$$

Round $\sqrt{|\mathbf{k}_1|}$ to 2.

Show all your work!