Cosine Similarity

Implement normalized cosine similarity to evaluate the embedding model.

Chapter Goals:

- Learn about cosine similarity and how it's used to compare embedding vectors
- Create a function that computes cosine similarities for a given word

A. Vector comparison

In mathematics, the standard way for comparing vector similarity is through *cosine similarity*. Since word embeddings are just vectors of real numbers, we can use also cosine similarity to compare embeddings for different words.

For two vectors, **u** and **v**, the equation for cosine similarity is

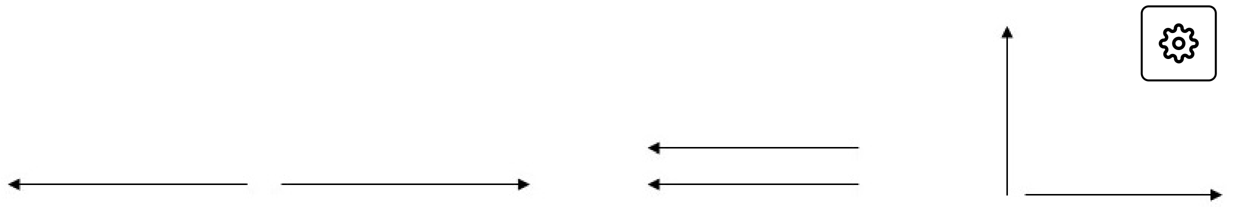
$$\text{cos sim} = \frac{\mathbf{u}}{||\mathbf{u}||_2} \cdot \frac{\mathbf{v}}{||\mathbf{v}||_2}$$

where $||v||_2$ represents the L2-norm (http://mathworld.wolfram.com/L2-Norm.html) of vector v, and $\cdot$ represents the dot product operation.

We refer to the quantity $\frac{v}{||v||_2}$ as the L2-normalization of vector v.

B. Correlation

The cosine similarity measures the *correlation* between two vectors, i.e. how closely related the two vectors are. The range of values for cosine similarity is [-1, 1]. A value of 1 means the vectors are perfectly identical, a value of -1 means the vectors are complete opposites, and a value of 0 means the vectors are *orthogonal* (i.e. completely uncorrelated).



Vector pairs showcasing cosine similarities of -1 (left), 1 (center), and 0 (right).

Note that the cosine similarity values are based on a spectrum, so we can measure correlation based on the cosine similarity's proximity to 1, 0, or -1. For example, we would expect the word embeddings for "orange" and "juice" to have a cosine similarity close to 1, since they are often used in the same context in conjunction with one another. On the other hand, we would expect "good" and "bad" to have a negative cosine similarity, since they are antonyms. And in most text corpuses, "chocolate" and "fence" would have a cosine similarity near 0, since they tend to be unrelated.

For example, imagine two vectors v_1 and v_2.

$$v_1 = \begin{bmatrix} 2 \\ 0 \\ 6 \end{bmatrix}, v_2 = \begin{bmatrix} 4 \\ 2 \\ 5 \end{bmatrix}$$

The L2 norm of v_1 is $\sqrt{2^2 + 0^2 + 6^2} = 6.324$

The L2 norm of v_2 is $\sqrt{4^2 + 2^2 + 5^2} = 6.708$

$$\frac{v_1}{6.324} = \begin{bmatrix} 0.316 \\ 0 \\ 0.949 \end{bmatrix}$$

$$\frac{v_2}{6.708} = \begin{bmatrix} 0.596 \\ 0.298 \\ 0.745 \end{bmatrix}$$

$$\begin{bmatrix} 0.316 \\ 0 \\ 0.949 \end{bmatrix} \cdot \begin{bmatrix} 0.596 \\ 0.298 \\ 0.745 \end{bmatrix} = 0.895$$

This number is very close to one, which means that v_1 and v_2 are very similar vectors

Time to Code!

In this chapter, you'll be completing the `compute_cos_sims` function, which computes cosine similarities between vocabulary words. Specifically, you'll be completing the function where it currently has the placeholder **"CODE HERE"**.

In order for the cosine similarities to be between 0 and 1, we need to first normalize both our retrieved embedding vector and the embedding matrix.

Set `normalized_embedding` equal to `tf.nn.l2_normalize` applied with `word_embedding` as the only argument.

Set `normalized_matrix` equal to `tf.nn.l2_normalize` applied with `self.embedding_matrix` as the required argument and `axis=1` as the keyword argument.

We can now calculate the embedding vector cosine similarities by matrix multiplying `normalized_embedding` and `normalized_matrix`. The matrix multiplication returns a vector with shape `(1, vocab_size)`, where each index contains a cosine similarity between the embedding vector for `word` and the embedding vector for the vocabulary word whose ID matches the index.

Set `cos_sims` equal to `tf.matmul` applied with `normalized_embedding` and `normalized_matrix` as the required arguments, and `transpose_b=True` as the keyword argument. Then return `cos_sims`.

```
import tensorflow as tf

# Skip-gram embedding model
class EmbeddingModel(object):
    # Model Initialization
    def __init__(self, vocab_size, embedding_dim):
        self.vocab_size = vocab_size
        self.embedding_dim = embedding_dim
        self.tokenizer = tf.keras.preprocessing.text.Tokenizer(num_words=

    # Forward run of the embedding model to retrieve embeddings
    def forward(self, target_ids):
        initial_bounds = 0.5 / self.embedding_dim
        initializer = tf.random_uniform(
            [self.vocab_size, self.embedding_dim],
            minval=-initial_bounds,
            maxval=initial_bounds)
        self.embedding_matrix = tf.get_variable('embedding_matrix',
            initializer=initializer)
        embeddings = tf.nn.embedding_lookup(self.embedding_matrix, target_
        return embeddings

    # Compute cosine similarites between the word's embedding
    # and all other embeddings for each vocabulary word
    def compute_cos_sims(self, word, training_texts):
        self.tokenizer.fit_on_texts(training_texts)
        word_id = self.tokenizer.word_index[word]
        word_embedding = self.forward([word_id])
        # CODE HERE
```

Solution

```
def compute_cos_sims(self, word, training_texts):
    self.tokenizer.fit_on_texts(training_texts)
    word_id = self.tokenizer.word_index[word]
    word_embedding = self.forward([word_id])
    normalized_embedding = tf.nn.l2_normalize(word_embedding)
    normalized_matrix = tf.nn.l2_normalize(self.embedding_matrix, axis=
    cos_sims = tf.matmul(normalized_embedding, normalized_matrix,
        transpose_b=True)
    return cos_sims

```

← Back
Embedding Loss

Next →
K-Nearest Neighbors

✓ Completed
