K-Nearest Neighbors

In this chapter, we will be learning how to calculate K-nearest neighbors based on cosine similarity.

Chapter Goals:

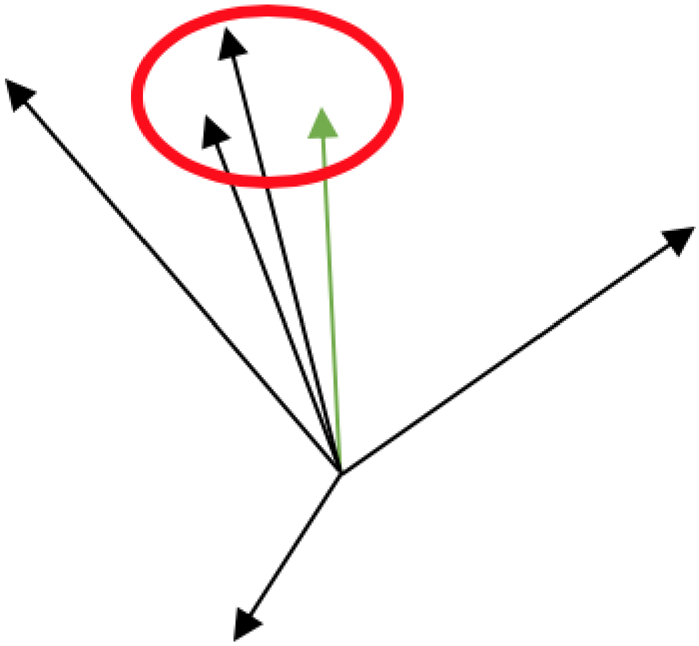
- Learn about K-nearest neighbors in terms of word similarity
- Create a function that computes the K-nearest neighbors for a given word

A. Similar words

When comparing cosine similarities for word embeddings, a common procedure is to find the *K-nearest neighbors* for a given word. This means that for a given a word w and an integer K, we can find the K vocabulary words whose embedding vectors have the highest cosine similarity to the embedding vector for w.

Using K-nearest neighbors can help us evaluate our embedding model and make sure it was trained properly. For example, if we notice that the K-nearest neighbors for any given word is always the same K words, then something may have gone wrong with our embedding training. Likewise, if the K-nearest neighbors are completely different from what we would expect to see (e.g. if the 3 nearest neighbors for the word "computer" are "waterfall", "ocean", and "soda"), then we may want to take a closer look at our model's embedding matrix.

When the embedding model is trained well, the K-nearest neighbors metric can provide some useful insights about the text corpus, particularly for specialized text corpora. For example, we normally expect the word "code" to be related to terms about computer programming or software. However, if our training corpus were related to military operations, the K-nearest neighbors for "code" might include words like "signal", "transmission", or "decode".



Circled in red are the K=2 nearest neighbors of the green vector. The nearest neighbors have the highest cosine similarity, meaning the directions they point in are nearest the green vector's direction

Time to Code!

In this chapter, you'll be completing the `k_nearest_neighbors` function, which computes the K-nearest neighbors for an input `word` using the TensorFlow utility function `tf.math.top_k`.

To find the K-nearest neighbors for `word`, we need to compute the cosine similarities between the embedding vectors for `word` and every other vocabulary word.

Set `cos_sims` equal to `self.compute_cos_sims` applied with `word` and `training_texts` as the arguments.

The returned `cos_sims` has shape `(1, self.vocab_size)`. However, when calculating the K-nearest neighbors, the extra dimension of size `1` is unnecessary. We can remove it using `tf.squeeze`.

Set `squeezed_cos_sims` equal to `tf.squeeze` applied with `cos_sims` as the only argument.

Now we can retrieve the K-nearest neighbors for `word`. The specific number of neighbors we retrieve is given by the integer argument, `k`.

Set `top_k_output` equal to `tf.math.top_k` applied with `squeezed_cos_sims` and `k` as the two arguments. Then return `top_k_output`.

```
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=self.vocab_size)

    # 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_ids)
        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])
        normalized_embedding = tf.nn.l2_normalize(word_embedding)
        normalized_matrix = tf.nn.l2_normalize(self.embedding_matrix, axis=1)
        cos_sims = tf.matmul(normalized_embedding, normalized_matrix,
            transpose_b=True)
        return cos_sims

    # Compute K-nearest neighbors for input word
    def k_nearest_neighbors(self, word, k, training_texts):
        # CODE HERE
        pass
```

Note that the output `top_k_output` is a tuple. The first element is the top K cosine similarities, while the second element is the actual word IDs corresponding to the top K nearest neighbors.