Skip-gram

Understand the difference between skip-gram and CBOW embedding models.

Chapter Goals:

- Learn about the types of models used to create embeddings
- Understand the difference between the skip-gram and CBOW models

A. Embedding models

In the previous chapter, we discussed how embedding vectors are based on target words and context windows. Specifically, we use these target words and context windows to create training *pairs*. Depending on the type of embedding model we use, the pairs will either be target-context pairs or context-target pairs.

Skip-gram

The skip-gram model uses target-context training pairs. Each pair has a target word as the first element and a context word as the second element. This means that every context window will produce multiple training pairs. For example, if the context window is "paul likes **singing** in french", (where **singing** is the target word), the training pairs would be

(singing, paul), (singing, likes)
(singing, in), (singing, french)

When training an embedding model, we convert the first element in each pair (for the skip-gram model, this is the target word) into its corresponding embedding vector. This will be discussed in greater depth in later chapters.

CBOW

The continuous-bag-of-words (CBOW) model uses context-target training pairs. Each pair will have all the context words as the first element and the target word as the second element. For example, if the context window is "tom

eats **spicy** crab salad" (where **spicy** is the target word), the training pair would be

((tom, eats, crab, salad), spicy)

Since the context element for the CBOW model contains multiple words, we use the average context embedding vector when training our embedding model. So in the example above, the context embedding vector would be the average between the four embedding vectors for "tom", "eats", "crab", and "salad".

B. Skip-gram vs. CBOW

When it comes to choosing between the skip-gram and CBOW embedding models, there are a few things to consider. Since the skip-gram model creates a training pair for each context word, it requires much less actual data than the CBOW model. However, this also means that the CBOW model is faster to train.

Furthermore, since the skip-gram model is creating multiple instances of training pairs for each target word, it can represent rarer words or phrases better than the CBOW model. On the other hand, the CBOW model provides more accurate embeddings for more common words.

In this course, the training data is small, so we will create a skip-gram model. However, when creating your own personal embedding model for an NLP task, it's best to take into account all the factors when making a decision.

Time to Code!

In this chapter, you'll be completing the `create_target_context_pairs` function, which creates target-context pairs for the skip-gram embedding model. Specifically, you'll be writing code in the portion of the function that says **"CODE HERE"**.

The target-context pairs we create will all be tuples containing `target_word` as the first element. The second element will be the context word ID at each context window index.

At each index, `j`, in

`range(left_incl, right_excl)`

where `j != i`, append the tuple

`(target_word, sequence[j])`

to the end of `pairs`.

```
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)

    def tokenize_text_corpus(self, texts):
        self.tokenizer.fit_on_texts(texts)
        sequences = self.tokenizer.texts_to_sequences(texts)
        return sequences

    # Convert a list of text strings into word sequences
    def get_target_and_context(self, sequence, target_index, window_size):
        target_word = sequence[target_index]
        half_window_size = window_size // 2
        left_incl = max(0, target_index - half_window_size)
        right_excl = min(len(sequence), target_index + half_window_size + 1)
        return target_word, left_incl, right_excl

    # Create (target, context) pairs for a given window size
    def create_target_context_pairs(self, texts, window_size):
        pairs = []
        sequences = self.tokenize_text_corpus(texts)
        for sequence in sequences:
            for i in range(len(sequence)):
                target_word, left_incl, right_excl = self.get_target_and_context(
                    sequence, i, window_size)
                # CODE HERE
        return pairs
```