Candidate Sampling

Understand why candidate sampling is used for embedding training.

Chapter Goals:

- Learn about candidate sampling and why it is useful for embedding training

A. Large vocabularies

To obtain good word embeddings, it is usually necessary to train an embedding model on a large amount of text data. This means that the vocabulary size will likely be very large, often reaching tens of thousands of words. However, having a large vocabulary size can significantly slow down training.

Training an embedding model is equivalent to multiclass classification, where the possible classes include every single vocabulary word. This means that we would need to calculate a softmax loss across every single vocabulary word during training, which can be incredibly time consuming for large vocabularies.

In order to mitigate the costly full softmax operation, we apply something called *candidate sampling*. With candidate sampling, we choose a much smaller fraction of the possible classes (i.e. vocabulary words) for computing the loss. This speeds up training significantly while also maintaining performance if we use the proper candidate samplers and loss function (more on this in the next chapter).

B. Computing logits

When we calculate the loss for an embedding model, we need to first compute the model's logits. We do this by setting up trainable weights and bias terms, which will be variables created by the `tf.get_variable` function.

Similar to the final layer of a multilayer perceptron, which computes the

MLP's logits, the weights and bias variables for the embedding model also

compute the logits, which are then converted into the loss based on the loss function.

Time to Code!

In this chapter, you'll be completing the `get_bias_weights` function, which gets the bias and weights for calculating the embedding loss (next chapter).

In order to calculate the loss for our candidate sampling algorithm, we need to create weight and bias variables. The weight variable will have shape `[self.vocab_size, self.embedding_dim]`, while the bias variable will have shape `[self.vocab_size]`.

We'll initialize the values for both the weight and bias variables to all 0's.

Set `weights_initializer` equal to `tf.zeros` applied with the weight variable's shape as the only argument.

Set `bias_initializer` equal to `tf.zeros` applied with the bias variable's shape as the only argument.

Next we'll create the weight and bias variables using these initializers and the `tf.get_variable` function.

Set `weights` equal to `tf.get_variable` with `'weights'` as the required argument `weights_initializer` as the `initializer` keyword argument.

Set `bias` equal to `tf.get_variable` with `'bias'` as the required argument `bias_initializer` as the `initializer` keyword argument.

Return a tuple with `weights` as the first element and `bias` as the second element.

```
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)
```

```
    # Get bias and weights for calculating loss
    def get_bias_weights(self):

        # CODE HERE
        pass
```