Embedding Loss

Learn about different loss functions for embedding training.

Chapter Goals:

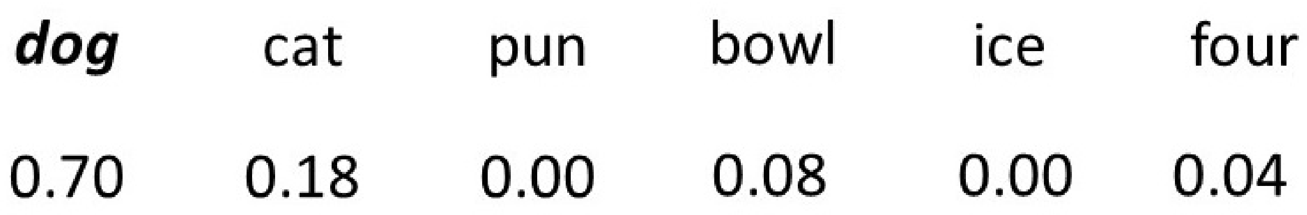
- Learn about the different types of candidate sampling algorithms and loss functions
- Calculate the embedding model's loss with the NCE loss function

A. Loss functions

As mentioned in the previous chapter, candidate sampling avoids performing a costly full softmax operation to calculate the embedding loss. Instead, there are two main loss functions we use: sampled softmax and NCE loss.

Sampled Softmax

As the name suggests, this is just a softmax loss with "sampled" classes. The classes we use to calculate the softmax include the actual context vocabulary word (the true label), as well as a randomly chosen set of words from the entire vocabulary to act as the false labels. In TensorFlow, we can compute the sampled softmax loss using the `tf.nn.sampled_softmax_loss` function.



dog	cat	pun	bowl	ice	four
0.70	0.18	0.00	0.08	0.00	0.04

Sampled softmax example for target word "growl". The actual context word (dog) and sampled words are shown with their corresponding softmax probabilities underneath.

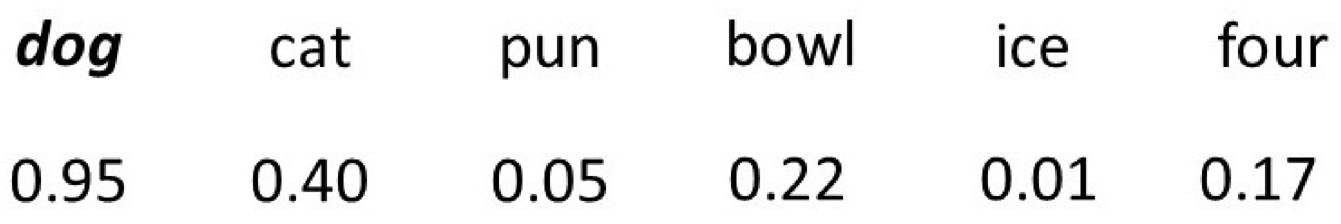
For training purposes, the sampled softmax loss gives a good approximation of the full softmax loss, while taking a fraction of the time to calculate. However, when evaluating a trained model's loss on a validation or test set, it's best to use the full softmax cross entropy loss for the most accurate metrics.

NCE Loss

The NCE (noise-contrastive estimation) loss takes a different approach than the sampled softmax. The idea is to convert the multiclass classification problem into a binary classification problem.

With the softmax-based loss functions, the embedding model is trained to predict the singular correct context word out of an entire set of vocabulary words. This can be a difficult task for the model to perform, even with a smaller sampled vocabulary set.

With noise-contrastive estimation, the model instead uses a sigmoid-based probabilistic approach. So for each of the sampled words, the model would want to output a low probability that the sampled word is part of the target word's context. On the other hand, the model should output a high probability for the true context word.



dog	cat	pun	bowl	ice	four
0.95	0.40	0.05	0.22	0.01	0.17

NCE probability example, using the same target, context, and sampled words as the previous example. Note that the probabilities don't sum to 1, since the sigmoid probabilities are independent of one another.

This approach is a much simpler task than calculating softmax probabilities, and is therefore normally favored when training an embedding model. In TensorFlow, we can compute the NCE loss using the `tf.nn.nce_loss` function.

Similar to the sampled softmax loss, the NCE loss provides a good loss approximation during training, but should be replaced by the full sigmoid cross entropy loss during evaluation or testing for the most accurate metrics.

Time to Code!

In this chapter, you'll be completing the `calculate_loss` function, which calculates the embedding loss.

To calculate the loss for our embedding model, we'll use the NCE loss function

To calculate the loss for our embedding model, we'll use the NCE loss function, `tf.nn.nce_loss`. The function takes in six required arguments:

- **weights:** The weights for computing the model's logits.
- **biases:** The bias terms for computing the model's logits.
- **labels:** The training batch's labels. For our skip-gram model, this will be the context IDs.
- **inputs:** The input embeddings for the NCE loss. For our skip-gram model, these are the target ID embeddings.
- **num_sampled:** The number of negative classes to sample.
- **num_classes:** The number of classes for calculating the loss. For embedding models, this is the vocabulary size.

The function's output is the NCE loss for each target-context pair in the training batch.

Set `nce_losses` equal to `tf.nn.nce_loss` applied with `weights`, `bias`, `context_ids`, `embeddings`, `num_negative_samples`, and `self.vocab_size` as the six required arguments.

We aggregate the NCE losses for the training batch into the overall loss, by taking the average across each of the NCE losses.

Set `overall_loss` equal to `tf.reduce_mean` applied to `nce_losses`. Then return `overall_loss`.

```
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):
        weights_initializer = tf.zeros([self.vocab_size, self.embedding_dim])
        bias_initializer = tf.zeros([self.vocab_size])
        weights = tf.get_variable('weights',
            initializer=weights_initializer)
        bias = tf.get_variable('bias',
            initializer=bias_initializer)
        return weights, bias

    # Calculate NCE Loss based on the retrieved embedding and context
    def calculate_loss(self, embeddings, context_ids, num_negative_samples):
        weights, bias = self.get_bias_weights()
        # CODE HERE
```