Training Data

Understand how training data is processed for sequence to sequence models.

Chapter Goals:

- Learn about the data used to train a seq2seq model
- Process input and output sequences into training data

A. Training task

For a seq2seq model, we use training pairs that contain an input sequence and an output sequence. For example, in machine translation the input sequence would be a sentence in one language, and the output sequence would be the sentence's correct translation in the other language.

During training, we perform two tasks:

1. **Input Task:** Extract useful information from the input sequence
2. **Output Task:** Calculate word probabilities at each output time step, using information from the input sequence and **previous words** in the output sequence

The input task is very common in NLP (e.g. text classification), and doesn't require any processing of the input sequence. The output task is equivalent to the task of language modeling, so it requires processing of the output sequence.

Specifically, we process the output sequence into two separate sequences: the ground truth sequence and the final token sequence.

B. Processing the output

The ground truth sequence for a seq2seq model is equivalent to the input sequence for a language model. It represents sequence prefixes that we use to calculate word probabilities at each time step of the output.

The final token sequence for a seq2seq model is equivalent to the output sequence when training a language model. It represents the "correct" words that the model should predict, based on the prefixes in the ground truth sequence.

Training Pair: ("he eats bread", "il mange du pain")

Ground Truth Sequence: ["il", "mange", "du"]

Final Token Sequence: ["mange", "du", "pain"]

Ground truth and final token sequences for an English-French machine translation model.

The ground truth and final token sequences are used to train a seq2seq model similar to how a language model is trained. The specific training details will be covered in later chapters when we introduce the encoder-decoder model.

C. SOS and EOS tokens

For seq2seq models, it is important to have start-of-sequence (SOS) and end-of-sequence (EOS) tokens. These tokens mark the start and end of a tokenized text sequence.

Input Sequence: ["SOS", "he", "eats", "bread", "EOS"]

Output Sequence: ["SOS", "il", "mange", "du", "pain", "EOS"]

Ground Truth Sequence: ["SOS", "il", "mange", "du"]

Final Token Sequence: ["mange", "du", "pain", "EOS"]

SOS and EOS tokens added to the training sequences from the previous example.

We will cover the reasoning behind using SOS and EOS tokens in later chapters. For now, just know that when the two tokens are added to the text corpus vocabulary, we refer to the new vocabulary as the *extended vocabulary*.

Time to Code!

In this section of the course you'll be creating a `Seq2SeqModel` object, which represents a seq2seq model.

Specifically, in this chapter you'll be completing the `make_training_tuple` function, which converts an input/output sequence pair into a training tuple.

The function is already filled with code that creates truncated versions of the output sequence and sets singleton lists for the SOS and EOS tokens. Your task is to use these truncated sequences and SOS/EOS tokens to create a tuple containing the update input, ground truth, and final token sequences.

For training the seq2seq model, the input sequence needs to have both the SOS and EOS tokens.

Set `input_sequence` equal to `input_sequence` with `sos_token` added to the front and `eos_token` added to the back.

The ground truth sequence is truncated in the back, so we only add the SOS token.

Set `ground_truth` equal to `truncate_back` with `sos_token` added to the front.

The final token sequence for the seq2seq model is truncated in the front, so we only add the EOS token.

Set `final_sequence` equal to `truncate_front` with `eos_token` added to the back.

Return a tuple containing `input_sequence` as the first element, `ground_truth` as the second, and `final_sequence` as the third.

```
import tensorflow as tf
tf_fc = tf.contrib.feature_column
tf_s2s = tf.contrib.seq2seq

# Seq2seq model
class Seq2SeqModel(object):
    def __init__(self, vocab_size, num_lstm_layers, num_lstm_units):
        self.vocab_size = vocab_size
        # Extended vocabulary includes start, stop token
        self.extended_vocab_size = vocab_size + 2
        self.num_lstm_layers = num_lstm_layers
        self.num_lstm_units = num_lstm_units
        self.tokenizer = tf.keras.preprocessing.text.Tokenizer(
            num_words=vocab_size)

    # Create a sequence training tuple from input/output sequences
    def make_training_tuple(self, input_sequence, output_sequence):
        truncate_front = output_sequence[1:]
        truncate_back = output_sequence[:-1]
        sos_token = [self.vocab_size]
        eos_token = [self.vocab_size + 1]
        # CODE HERE
```