# Padding

Understand the purpose of padding with respect to tokenized sequences.

Chapter Goals:

- Learn about sequence lengths and padding

## A. Varied length sequence

When dealing with most neural networks, the input data always has a fixed length. This is because most neural networks have what's known as a *feed-forward* structure, meaning that they utilize multiple layers of fixed sizes to compute the network's output.

However, since text data involves different sized text sequences (e.g. sentences, passages, etc.), our language model needs to be able to handle input data of varied lengths. Therefore, we use a *recurrent* neural network (discussed in greater detail in upcoming chapters) for the basis of our language model.

## B. Padded sequences

While the recurrent neural network architecture allows the language model to take in different length input texts, we still need each tokenized text sequence in a training batch to be the same length. This is because a training batch must have proper tensor shape, which in this case means it needs to be a 2-D matrix.

The way we mimic a proper 2-D matrix shape with sequences of varied lengths is to apply *padding*. For each sequence that's shorter than the maximum sequence length, we append a special non-vocabulary token to the end of the sequence until its length is equal to the maximum sequence length. Usually the special padding token is given the ID 0, while each vocabulary word ID is a positive integer.

$$[1, 5, 10, 9, 0, 0, 0]$$

$$[1, 5, 10, 9, 0, 0, 0]$$

$$[12, 2, 3, 4, 17, 8, 2]$$
$$[3, 6, 1, 0, 0, 0, 0]$$

$$[9, 14, 9, 16, 17, 0, 0]$$

A batch of 4 input sequences for a language model. The maximum sequence length is 7, so each sequence is padded with 0's until its length is 7.

## Time to Code!

In this chapter, you'll be completing another helper function for the `get_input_target_sequence` function.

The helper function you'll complete is the `pad_sequences` function. This function is used when the length of `sequence` is less than `self.max_length`.

When the length of `sequence` is less than `self.max_length`, we need to add padding. The amount of padding is just the difference between the maximum sequence length and the length of `sequence`.

**Inside the `pad_sequences` function, set `padding_amount` equal to `max_length` minus the length of `sequence`.**

The `Tokenizer` object never uses `0` as an ID corresponding to a corpus word. This helps us a lot, since we can now safely use `0` as our padding token without worrying about conflicts.

**Set `padding` equal to a list of all `0`, whose length is `padding_amount`.**

As in the previous chapter, `input_sequence` represents the input sequence with the final token removed, while `target_sequence` represents the target sequence with the first token removed.

The difference here is that both the input and target sequences will be padded up to `max_length`.

**Set `input_sequence` equal to `sequence[:-1]` with `padding` added to the end. Also set `target_sequence` equal to `sequence[1:]` with `padding` added to the end.**

**Return a tuple with `input_sequence` as the first element and `target_sequence` as the second element.**

`target_sequence` as the second element.

```
import tensorflow as tf

def pad_sequences(sequence, max_length):
    # CODE HERE
    pass

# LSTM Language Model
class LanguageModel(object):
    # Model Initialization
    def __init__(self, vocab_size, max_length, num_lstm_units, num_lstm_layers):
        self.vocab_size = vocab_size
        self.max_length = max_length
        self.num_lstm_units = num_lstm_units
        self.num_lstm_layers = num_lstm_layers
        self.tokenizer = tf.keras.preprocessing.text.Tokenizer(num_words=vocab_size)

    def get_input_target_sequence(self, sequence):
        seq_len = len(sequence)
        if seq_len >= self.max_length:
            input_sequence, target_sequence = truncate_sequences(
                sequence, self.max_length
            )
        else:
            input_sequence, target_sequence = pad_sequences(
                sequence, self.max_length
            )
        return input_sequence, target_sequence
```