Open in Colab

Generating Sequences and Padding

In this lab, you will look at converting your input sentences into a sequence of tokens. Similar to images in the previous course, you need to prepare text data with uniform size before feeding it to your model. You will see how to do these in the next sections.

Text to Sequences

In the previous lab, you saw how to generate a `word_index` dictionary to generate tokens for each word in your corpus. You can use then use the result to convert each of the input sentences into a sequence of tokens. That is done using the `texts_to_sequences()` method as shown below.

In [1]:
```
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

# Define your input texts
sentences = [
    'I love my dog',
    'I love my cat',
    'You love my dog!',
    'Do you think my dog is amazing?'
]

# Initialize the Tokenizer class
tokenizer = Tokenizer(num_words = 100, oov_token="<OOV>")

# Tokenize the input sentences
tokenizer.fit_on_texts(sentences)

# Get the word index dictionary
word_index = tokenizer.word_index

# Generate list of token sequences
sequences = tokenizer.texts_to_sequences(sentences)

# Print the result
print("\nWord Index = " , word_index)
print("\nSequences = " , sequences)
```

```
Word Index =  {'<OOV>': 1, 'my': 2, 'love': 3, 'dog': 4, 'i': 5, 'you': 6, 'cat': 7, 'do': 8, 'think': 9, 'is': 10, 'amazing': 11}

Sequences =  [[5, 3, 2, 4], [5, 3, 2, 7], [6, 3, 2, 4], [8, 6, 9, 2, 4, 10, 11]]
```

Padding

As mentioned in the lecture, you will usually need to pad the sequences into a uniform length because that is what your model expects. You can use the pad_sequences for that. By default, it will pad according to the length of the longest sequence. You can override this with the `maxlen` argument to define a specific length. Feel free to play with the other arguments shown in class and compare the result.

In [2]:
```
# Pad the sequences to a uniform length
padded = pad_sequences(sequences, maxlen=5)

# Print the result
print("\nPadded Sequences:")
print(padded)
```

```
Padded Sequences:
[[ 0  5  3  2  4]
 [ 0  5  3  2  7]
 [ 0  6  3  2  4]
 [ 9  2  4 10 11]]
```

Out-of-vocabulary tokens

Notice that you defined an `oov_token` when the `Tokenizer` was initialized earlier. This will be used when you have input words that are not found in the `word_index` dictionary. For example, you may decide to collect more text after your initial training and decide to not re-generate the `word_index`. You will see this in action in the cell below. Notice that the token `1` is inserted for words that are not found in the dictionary.

In [3]:
```
# Try with words that the tokenizer wasn't fit to
test_data = [
    'i really love my dog',
    'my dog loves my manatee'
]

# Generate the sequences
test_seq = tokenizer.texts_to_sequences(test_data)

# Print the word index dictionary
print("\nWord Index = " , word_index)

# Print the sequences with OOV
print("\nTest Sequence = ", test_seq)

# Print the padded result
padded = pad_sequences(test_seq, maxlen=10)
print("\nPadded Test Sequence: ")
print(padded)
```

```
Word Index =  {'<OOV>': 1, 'my': 2, 'love': 3, 'dog': 4, 'i': 5, 'you': 6, 'cat': 7, 'do': 8, 'think': 9, 'is': 10, 'amazing': 11}

Test Sequence =  [[5, 1, 3, 2, 4], [2, 4, 1, 2, 1]]

Padded Test Sequence:
[[0 0 0 0 0 5 1 3 2 4]
 [0 0 0 0 0 2 4 1 2 1]]
```

This concludes another introduction to text data preprocessing. So far, you've just been using dummy data. In the next exercise, you will be applying the same concepts to a real-world and much larger dataset.