Ungraded Lab: Tokenizing the Sarcasm Dataset

In this lab, you will apply what you've learned in the past two exercises to preprocess the News Headlines Dataset for Sarcasm Detection. This contains news headlines which are labeled as sarcastic or not. You will revisit this dataset in later labs so it is good to be acquainted with it now.

Imports

Let's start by importing the packages and methods you will use in this lab.

In [1]:
```
import tensorflow as tf
import json
import tensorflow_datasets as tfds
from tensorflow.keras.utils import pad_sequences
```

Download and inspect the dataset

Then, you will fetch the dataset and preview some of its elements.

In [2]:
```
# Download the dataset
!wget -nc https://storage.googleapis.com/tensorflow-1-public/course3/sarcasm.json
```

```
File ‘sarcasm.json’ already there; not retrieving.
```

The dataset is saved as a JSON file and you can use Python's `json` module to load it into your workspace. The cell below unpacks the JSON file into a list.

In [3]:
```
# Load the JSON file
with open("./sarcasm.json", 'r') as f:
    datastore = json.load(f)
```

You can inspect a few of the elements in the list. You will notice that each element consists of a dictionary with a URL link, the actual headline, and a label named `is_sarcastic`. Printed below are two elements with contrasting labels.

In [4]:
```
# Non-sarcastic headline
print(datastore[0])

# Sarcastic headline
print(datastore[20000])
```

```
{'article_link': 'https://www.huffingtonpost.com/entry/versace-black-code_us_5861fbefe4b0de3a08f600d5', 'headline': "former versace stor
e clerk sues over secret 'black code' for minority shoppers", 'is_sarcastic': 0}
{'article_link': 'https://www.theonion.com/pediatricians-announce-2011-newborns-are-ugliest-babies-1819572977', 'headline': 'pediatricia
ns announce 2011 newborns are ugliest babies in 30 years', 'is_sarcastic': 1}
```

With that, you can collect the headlines because those are the string inputs that you will preprocess into numeric features.

In [5]:
```
# Append the headline elements into the list
sentences = [item['headline'] for item in datastore]
```

Preprocessing the headlines

You can convert the sentences list above into padded sequences by using the same methods you've been using in the previous labs. The cells below will build the vocabulary, then use that to generate the list of post-padded sequences for each of the 26,709 headlines.

In [6]:
```
# Instantiate the layer
vectorize_layer = tf.keras.layers.TextVectorization()

# Build the vocabulary
vectorize_layer.adapt(sentences)

# Apply the layer for post padding
post_padded_sequences = vectorize_layer(sentences)
```

You can view the results for a particular headline by changing the value of `index` below.

In [7]:
```
# Print a sample headline and sequence
index = 2
print(f'sample headline: {sentences[index]}')
print(f'padded sequence: {post_padded_sequences[index]}')
print()

# Print dimensions of padded sequences
```

```
print(f'shape of padded sequences: {post_padded_sequences.shape}')
```

```
sample headline: mom starting to fear son's web series closest thing she will have to grandchild
padded sequence: [  140   825     2   813  1100  2048   571  5057   199   139    39    46
     2 13050     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0]

shape of padded sequences: (26709, 39)
```

For prepadding, you have to setup the `TextVectorization` layer differently. You don't want to have the automatic postpadding shown above, and instead have sequences with variable length. Then, you will pass it to the `pad_sequences()` utility function you used in the previous lab. The cells below show one way to do it:

- First, you will initialize the `TextVectorization` layer and set its `ragged` flag to `True`. This will result in a ragged tensor which simply means a tensor with variable-length elements. The sequences will indeed have different lengths after removing the zeroes, thus you will need the ragged tensor to contain them.
- Like before, you will use the layer's `adapt()` method to generate a vocabulary.
- Then, you will apply the layer to the string sentences to generate the integer sequences. As mentioned, this will not be post-padded.
- Lastly, you will pass this ragged tensor to the pad_sequences() function to generate pre-padded sequences.

In [8]:
```
# Instantiate the layer and set the `ragged` flag to `True`
vectorize_layer = tf.keras.layers.TextVectorization(ragged=True)

# Build the vocabulary
vectorize_layer.adapt(sentences)
```

In [9]:
```
# Apply the layer to generate a ragged tensor
ragged_sequences = vectorize_layer(sentences)
```

In [10]:
```
# Print a sample headline and sequence
index = 2
print(f'sample headline: {sentences[index]}')
print(f'padded sequence: {ragged_sequences[index]}')
print()

# Print dimensions of padded sequences
print(f'shape of padded sequences: {ragged_sequences.shape}')
```

```
sample headline: mom starting to fear son's web series closest thing she will have to grandchild
padded sequence: [  140   825     2   813  1100  2048   571  5057   199   139    39    46
     2 13050]

shape of padded sequences: (26709, None)
```

In [11]:
```
# Apply pre-padding to the ragged tensor
pre_padded_sequences = pad_sequences(ragged_sequences.numpy())

# Preview the result for the 2nd sequence
pre_padded_sequences[2]
```

Out[11]:
```
array([    0,     0,     0,     0,     0,     0,     0,     0,     0,
           0,     0,     0,     0,     0,     0,     0,     0,     0,
           0,     0,     0,     0,     0,     0,     0,   140,   825,
           2,   813,  1100,  2048,   571,  5057,   199,   139,    39,
          46,     2, 13050], dtype=int32)
```

You can see the results for post-padded and pre-padded sequences by changing the value of `index` below.

In [12]:
```
# Print a sample headline and sequence
index = 2
print(f'sample headline: {sentences[index]}')
print()
print(f'post-padded sequence: {post_padded_sequences[index]}')
print()
print(f'pre-padded sequence: {pre_padded_sequences[index]}')
print()

# Print dimensions of padded sequences
print(f'shape of post-padded sequences: {post_padded_sequences.shape}')
print(f'shape of pre-padded sequences: {pre_padded_sequences.shape}')
```

```
sample headline: mom starting to fear son's web series closest thing she will have to grandchild

post-padded sequence: [  140   825     2   813  1100  2048   571  5057   199   139    39    46
     2 13050     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0]

pre-padded sequence: [    0     0     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0   140   825     2   813  1100  2048   571  5057   199   139    39
    46     2 13050]

shape of post-padded sequences: (26709, 39)
shape of pre-padded sequences: (26709, 39)
```

This concludes the short demo on text data preprocessing on a relatively large dataset. Next week, you will start building models that can be trained on these output sequences. See you there!