AI for Medicine Course 3 Week 2 lecture notebook

Preparing Input for Text Classification

In this lecture notebook you'll be working with input for text classification models. You'll simulate BERT's tokenizer for a simple example, and then use it for real in the upcoming assignment!

Import Library

In [1]:
```
import tensorflow as tf
```

Define Model Inputs

Say you're in the following situation: You have a passage containing a patient's medical information and would like your model to be able to answer questions using information from this passage. First, you'll need to reformulate this question and text in a way that BERT can interpret correctly. Let's define a question and the passage:

In [2]:
```
q = "How old is the patient?"
p = '''
The patient is a 64-year-old male named Bob.
He has no history of chronic spine conditions but is
showing mild degenerative changes in the lumbar spine and old right rib fractures.
'''
```

Tokenize Sentences

With this information, you would normally use BERT's tokenizer to tokenize the sentences like this:

```
tokenizer.tokenize(q)
```

Luckily, this has already been taken care of for you!

In [3]:
```
q_tokens = ['How', 'old', 'is', 'the', 'patient', '?']
p_tokens = ['The', 'patient', 'is', 'a', '64', 'year', 'old', 'male', 'named', 'Bob', '.',
            'He', 'has', 'no', 'history', 'of', 'chronic', 'spine', 'conditions', 'but', 'is',
            'showing', 'mild', 'de', '##gene', '##rative', 'changes', 'in', 'the', 'l', '##umba',
            '##r', 'spine', 'and', 'old', 'right', 'rib', 'fracture', '##s', '.']
classification_token = '[CLS]'
separator_token = '[SEP]'
```

The classification token and separator token are also provided. These tokens can be accessed using the tokenizer like so:

```
CLS = tokenizer.cls_token
SEP = tokenizer.sep_token
```

These tokens are really important because you'll need to combine the question and passage tokens into a single list of tokens. These special tokens allow BERT to understand which is which.

The CLS, or classification token, should come first. Then, use the SEP token as a separator between the question and the passage.

In [4]:
```
tokens = []
tokens.append(classification_token)
tokens.extend(q_tokens)
tokens.append(separator_token)
tokens.extend(p_tokens)
print(f"The token list looks like this: \n\n{tokens}")
```

```
The token list looks like this:

['[CLS]', 'How', 'old', 'is', 'the', 'patient', '?', '[SEP]', 'The', 'patient', 'is', 'a', '64', 'year', 'old', 'male', 'nam
ed', 'Bob', '.', 'He', 'has', 'no', 'history', 'of', 'chronic', 'spine', 'conditions', 'but', 'is', 'showing', 'mild', 'de',
'##gene', '##rative', 'changes', 'in', 'the', 'l', '##umba', '##r', 'spine', 'and', 'old', 'right', 'rib', 'fracture', '##
s', '.']
```

Convert Tokens to Numerical Representations

You now have the complete token list. However, you still need to convert these tokens into numeric representations of themselves. Usually, you would convert them like this:

```
tokenizer.convert_tokens_to_ids(tokens)
```

Fortunately for you, this has also been provided:

In [5]:
```
token_ids = [101, 1731, 1385, 1110, 1103, 5351, 136, 102, 1109, 5351, 1110, 170, 3324,
             1214, 1385, 2581, 1417, 3162, 119, 1124, 1144, 1185, 1607, 1104, 13306, 8340,
             2975, 1133, 1110, 4000, 10496, 1260, 27054, 15306, 2607, 1107, 1103, 181, 25509,
             1197, 8340, 1105, 1385, 1268, 23298, 22869, 1116, 119]
```

Apply Padding

This is great, except the length of the list of `token ids` depends on the number of words in the question. The passage and BERT only accepts fixed-size input.

To deal with this, you'll use **padding**, which involves filling out the rest of this list with an empty value until it reaches a maximum length that you set.

In this case you'll use "0" as your empty value, 60 as the maximum length, and then leverage the `pad_sequences()` function from Keras' Sequence module:

In [6]:
```
from tensorflow.keras.preprocessing.sequence import pad_sequences

max_length = 60
```

```
token_ids = pad_sequences([token_ids], padding="post", maxlen=max_length)
token_ids
```

Out[6]:
```
array([[  101,  1731,  1385,  1110,  1103,  5351,   136,   102,  1109,
         5351,  1110,   170,  3324,  1214,  1385,  2581,  1417,  3162,
          119,  1124,  1144,  1185,  1607,  1104, 13306,  8340,  2975,
         1133,  1110,  4000, 10496,  1260, 27054, 15306,  2607,  1107,
         1103,   181, 25509,  1197,  8340,  1105,  1385,  1268, 23298,
        22869,  1116,   119,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0]], dtype=int32)
```

It appears the padding has been done correctly. Usually, this list of token ids would need to be a Tensor, but this is easily recast using the `convert_to_tensor()` function from TensorFlow:

In [7]:
```
token_ids = tf.convert_to_tensor(token_ids)
token_ids
```

Out[7]:
```
<tf.Tensor: id=0, shape=(1, 60), dtype=int32, numpy=
array([[  101,  1731,  1385,  1110,  1103,  5351,   136,   102,  1109,
         5351,  1110,   170,  3324,  1214,  1385,  2581,  1417,  3162,
          119,  1124,  1144,  1185,  1607,  1104, 13306,  8340,  2975,
         1133,  1110,  4000, 10496,  1260, 27054, 15306,  2607,  1107,
         1103,   181, 25509,  1197,  8340,  1105,  1385,  1268, 23298,
        22869,  1116,   119,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0]], dtype=int32)>
```

Add the Input Mask

You're almost done! BERT still needs an input mask as one of its inputs. An input mask is just a list of the same length as the `token ids` list, indicating whether a certain position contains a token or empty values created from padding.

You'll see how to do this using Keras' Masking layer, but in this case it could be done more simply using a little Python.

If you're interested in learning some of the details of padding and masking, check this out.

In [8]:
```
from tensorflow.keras import layers

masking_layer = layers.Masking()

unmasked = tf.cast(
    tf.tile(tf.expand_dims(tf.convert_to_tensor(
        token_ids), axis=-1), [1, 1, 1]),
    tf.float32)

masked = masking_layer(unmasked)
token_mask = masked._keras_mask
token_mask
```

Out[8]:
```
<tf.Tensor: id=13, shape=(1, 60), dtype=bool, numpy=
array([[ True,  True,  True,  True,  True,  True,  True,  True,  True,
         True,  True,  True,  True,  True,  True,  True,  True,  True,
         True,  True,  True,  True,  True,  True,  True,  True,  True,
         True,  True,  True,  True,  True,  True,  True,  True,  True,
         True,  True,  True,  True,  True,  True,  True,  True,  True,
         True,  True,  True, False, False, False, False, False, False,
        False, False, False, False, False, False]])>
```

As you can see, the token mask outputs `True` for tokens and `False` for padding.

Now you've successfully created and formatted the inputs necessary to use the BERT model!

In cases where you don't want to use Keras, you can get the same result by using plain Python lists, which provide a different structure and data type than the one produced by the Masking layer.

Manipulating Tensors

Before moving on, let's convert the padded token ids list to the same type as the one you just did:

In [9]:
```
padded_token_ids = [101,  1731,  1385,  1110,  1103,  5351,   136,   102,  1109,
                    5351,  1110,   170,  3324,  1214,  1385,  2581,  1417,  3162,
                    119,  1124,  1144,  1185,  1607,  1104, 13306,  8340,  2975,
                    1133,  1110,  4000, 10496,  1260, 27054, 15306,  2607,  1107,
                    1103,   181, 25509,  1197,  8340,  1105,  1385,  1268, 23298,
                    22869,  1116,   119,     0,     0,     0,     0,     0,     0,
                    0,     0,     0,     0,     0,     0]
```

First let's convert the list into a tensor:

In [10]:
```
padded_token_ids = tf.convert_to_tensor(padded_token_ids)
padded_token_ids
```

Out[10]:
```
<tf.Tensor: id=14, shape=(60,), dtype=int32, numpy=
array([  101,  1731,  1385,  1110,  1103,  5351,   136,   102,  1109,
         5351,  1110,   170,  3324,  1214,  1385,  2581,  1417,  3162,
          119,  1124,  1144,  1185,  1607,  1104, 13306,  8340,  2975,
         1133,  1110,  4000, 10496,  1260, 27054, 15306,  2607,  1107,
         1103,   181, 25509,  1197,  8340,  1105,  1385,  1268, 23298,
        22869,  1116,   119,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0], dtype=int32)>
```

Notice that the shape of this tensor doesn't match the desired one. You can easily check this by doing the following:

In [11]:
```
padded_token_ids.shape == token_ids.shape
```

Out[11]:
```
False
```

Using the `expand_dims()` function from TensorFlow you can reshape this tensor like this:

In [12]:
```
padded_token_ids = tf.expand_dims(padded_token_ids, 0)
padded_token_ids
```

Out[12]:
```
<tf.Tensor: id=16, shape=(1, 60), dtype=int32, numpy=
array([[  101,  1731,  1385,  1110,  1103,  5351,   136,   102,  1109,
         5351,  1110,   170,  3324,  1214,  1385,  2581,  1417,  3162,
          119,  1124,  1144,  1185,  1607,  1104, 13306,  8340,  2975,
         1133,  1110,  4000, 10496,  1260, 27054, 15306,  2607,  1107,
         1103,   181, 25509,  1197,  8340,  1105,  1385,  1268, 23298,
        22869,  1116,   119,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0]], dtype=int32)>
```

In [13]:
```
padded_token_ids.shape == token_ids.shape
```

Out[13]:
```
True
```

Congratulations on finishing this lecture notebook!!!

You're all done preparing some simple input for BERT. Excellent job!