Putting it all together (TensorFlow)

Install the Transformers, Datasets, and Evaluate libraries to run this notebook.

In [16]:

```
!pip install datasets evaluate transformers[sentencepiece]
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-w
heels/public/simple/
Requirement already satisfied: datasets in /usr/local/lib/python3.7/dist-packa
ges (2.4.0)
Requirement already satisfied: evaluate in /usr/local/lib/python3.7/dist-packa
ges (0.2.2)
Requirement already satisfied: transformers[sentencepiece] in /usr/local/lib/p
ython3.7/dist-packages (4.21.1)
Requirement already satisfied: tqdm>=4.62.1 in /usr/local/lib/python3.7/dist-p
ackages (from datasets) (4.64.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.7/dist-pack
ages (from datasets) (21.3)
Requirement already satisfied: pandas in /usr/local/lib/python3.7/dist-package
s (from datasets) (1.3.5)
Requirement already satisfied: dill<0.3.6 in /usr/local/lib/python3.7/dist-pac
kages (from datasets) (0.3.5.1)
Requirement already satisfied: fsspec[http]>=2021.11.1 in /usr/local/lib/pytho
n3.7/dist-packages (from datasets) (2022.7.1)
Requirement already satisfied: aiohttp in /usr/local/lib/python3.7/dist-packag
es (from datasets) (3.8.1)
Requirement already satisfied: multiprocess in /usr/local/lib/python3.7/dist-p
ackages (from datasets) (0.70.13)
Requirement already satisfied: importlib-metadata in /usr/local/lib/python3.7/
dist-packages (from datasets) (4.12.0)
Requirement already satisfied: requests>=2.19.0 in /usr/local/lib/python3.7/di
st-packages (from datasets) (2.23.0)
Requirement already satisfied: responses<0.19 in /usr/local/lib/python3.7/dist
-packages (from datasets) (0.18.0)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.7/dist-pa
ckages (from datasets) (1.21.6)
Requirement already satisfied: pyarrow>=6.0.0 in /usr/local/lib/python3.7/dist
-packages (from datasets) (6.0.1)
Requirement already satisfied: huggingface-hub<1.0.0,>=0.1.0 in /usr/local/li
b/python3.7/dist-packages (from datasets) (0.8.1)
Requirement already satisfied: xxhash in /usr/local/lib/python3.7/dist-package
s (from datasets) (3.0.0)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/py
thon3.7/dist-packages (from huggingface-hub<1.0.0,>=0.1.0->datasets) (4.1.1)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.7/dist-pa
ckages (from huggingface-hub<1.0.0,>=0.1.0->datasets) (6.0)
Requirement already satisfied: filelock in /usr/local/lib/python3.7/dist-packa
ges (from huggingface-hub<1.0.0,>=0.1.0->datasets) (3.8.0)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /usr/local/lib/pyth
on3.7/dist-packages (from packaging->datasets) (3.0.9)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/
dist-packages (from requests>=2.19.0->datasets) (2022.6.15)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-p
ackages (from requests>=2.19.0->datasets) (2.10)
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/d
ist-packages (from requests>=2.19.0->datasets) (3.0.4)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /us
r/local/lib/python3.7/dist-packages (from requests>=2.19.0->datasets) (1.25.1
1)
```

```
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3.
7/dist-packages (from aiohttp->datasets) (6.0.2)
```

Tokenize the input

The Transformer's inputs are sequences of *token identifiers* (of type integer)

- Need to convert text into tokens ("word parts")
- Need to convert the tokens to token identifiers

A *model* is identified by a **checkpoint**

- string identifying the model architecture and state at which training was ended
    - n.b., if you train for longer, the weights will change (resulting in a different checkpoint)

A pre-trained model is usually paired with the Tokenizer on which it was trained.

We can obtain the Tokenizer from a checkpoint via
`AutoTokenizer.from_pretrained(checkpoint)`

In [17]:
```
from transformers import AutoTokenizer

checkpoint = "distilbert-base-uncased-finetuned-sst-2-english"
tokenizer = AutoTokenizer.from_pretrained(checkpoint)
```

Let's understand the Tokenizer

In [18]:

```python
sequence = "I've been waiting for a HuggingFace course my whole life."

model_inputs = tokenizer(sequence)

print("Model inputs: ", model_inputs)

print("Model inputs (input_ids): ", model_inputs["input_ids"])
```

```
Model inputs:  {'input_ids': [101, 1045, 1005, 2310, 2042, 3403, 2005, 1037, 1
7662, 12172, 2607, 2026, 2878, 2166, 1012, 102], 'attention_mask': [1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]}
Model inputs (input_ids):  [101, 1045, 1005, 2310, 2042, 3403, 2005, 1037, 176
62, 12172, 2607, 2026, 2878, 2166, 1012, 102]
```

The `input_ids` key are the *token identifiers*.

Out of curiousity, we can obtain the token identifiers in 2 sub-steps

- convert text to tokens
- convert tokens to token identifiers

In [19]:
```
print("Text: ", sequence)
```
```
Text:  I've been waiting for a HuggingFace course my whole life.
```

```
In [20]: print("Text: ", sequence)

         print("\nFirst step: Manually convert sequence of characters to sequence of toke
         ns")
         tokens = tokenizer.tokenize(sequence)

         print("Tokens: ", tokens)

         print("\nSecond step: Manually convert tokens to ids")
         token_ids = tokenizer.convert_tokens_to_ids(tokens)

         print("Token identifiers: ", token_ids)

         # Verify that the sequence of token ids created manually is identical to that cr
         eated by the one-step process
         model_inputs = tokenizer(sequence)

         assert(token_ids == model_inputs["input_ids"][1:-1])
         print('\nVerified ! token_ids == model_inputs["input_ids"][1:-1]')
         print('\n\tThat is: model_inputs has bracketed the token_ids with the special st
         art and end tokens')

         print("\n")
         print("Decoded model inputs (input_ids): ", tokenizer.decode(model_inputs["input
         _ids"]))
         print("Decoded token identifiers: ", tokenizer.decode(token_ids) )
```

```
Text:  I've been waiting for a HuggingFace course my whole life.

First step: Manually convert sequence of characters to sequence of tokens
Tokens:  ['i', "'", 've', 'been', 'waiting', 'for', 'a', 'hugging', '##face',
'course', 'my', 'whole', 'life', '.']

Second step: Manually convert tokens to ids
Token identifiers:  [1045, 1005, 2310, 2042, 3403, 2005, 1037, 17662, 12172, 2
607, 2026, 2878, 2166, 1012]

Verified ! token_ids == model_inputs["input_ids"][1:-1]

        That is: model_inputs has bracketed the token_ids with the special sta
rt and end tokens


Decoded model inputs (input_ids):  [CLS] i've been waiting for a huggingface c
ourse my whole life. [SEP]
Decoded token identifiers:  i've been waiting for a huggingface course my whol
e life.
```

You can see that the

- `input_ids` has the special token `[CLS]` added at the start and `[SEP]` added at the end of the text
- These special tokens are required by the Transformer model

`token_ids` is identical to `input_ids` except for these special tokens

The Tokenizer's behavior can be modified.

When dealing with more than one example, the example lengths (after tokenization) may have different lengths.

The Tokenizer can adapt it's behavior.

We just list the behavior without going further into it.

In [21]:

```
# Will pad the sequences up to the maximum sequence length
model_inputs = tokenizer(sequence, padding="longest")

# Will pad the sequences up to the model max length
# (512 for BERT or DistilBERT)
model_inputs = tokenizer(sequence, padding="max_length")

# Will pad the sequences up to the specified max length
model_inputs = tokenizer(sequence, padding="max_length", max_length=8)
```

In [22]:
```
sequences = ["I've been waiting for a HuggingFace course my whole life.", "So ha
ve I!"]

# Will truncate the sequences that are longer than the model max length
# (512 for BERT or DistilBERT)
model_inputs = tokenizer(sequences, truncation=True)

# Will truncate the sequences that are longer than the specified max length
model_inputs = tokenizer(sequences, max_length=8, truncation=True)
```

In [23]:

```
import tensorflow as tf
from transformers import AutoTokenizer, TFAutoModelForSequenceClassification

checkpoint = "distilbert-base-uncased-finetuned-sst-2-english"
tokenizer = AutoTokenizer.from_pretrained(checkpoint)
model = TFAutoModelForSequenceClassification.from_pretrained(checkpoint)
sequences = ["I've been waiting for a HuggingFace course my whole life.", "So ha
ve I!"]

tokens = tokenizer(sequences, padding=True, truncation=True, return_tensors="t
f")
output = model(**tokens)
```

```
All model checkpoint layers were used when initializing TFDistilBertForSequenc
eClassification.

All the layers of TFDistilBertForSequenceClassification were initialized from
the model checkpoint at distilbert-base-uncased-finetuned-sst-2-english.
If your task is similar to the task the model of the checkpoint was trained o
n, you can already use TFDistilBertForSequenceClassification for predictions w
ithout further training.
```

In [24]:
```
output
```

Out[24]:
```
TFSequenceClassifierOutput(loss=None, logits=<tf.Tensor: shape=(2, 2), dtype=f
loat32, numpy=
array([[-1.5606955,  1.6122806],
       [-3.6183183,  3.9137495]], dtype=float32)>, hidden_states=None, attenti
ons=None)
```

The output is a `Tensor`

- they are the `logits` (scores, **not** probabilities) of the Binary Classification model

Convert them to probabilities

In [25]:

```
import numpy as np
probs = tf.nn.softmax(output["logits"]).numpy()

ex_classes = np.argmax(probs, axis=1)

for i, prob in enumerate(probs):
  ex_class = ex_classes[i]
  print(f"Example {i}: Class {ex_class:d} with probability {probs[i, ex_class]:
3.2f}")
```

```
Example 0: Class 1 with probability 0.96
Example 1: Class 1 with probability 1.00
```

Classifier model output type: logits vs probabilities

There is a **subtle but important** way to pass Loss function names into Keras when using HuggingFace.

Recall that some Classifiers, e.g., Logistic Regression, work by

- computing a score/logit
$$\text{logit} = \Theta \cdot \mathbf{x}$$
- converting the logit to a probability
    - by applying a `softmax` to the logits

Our practice has been to assume that

- the model output
$$\mathbf{y} = \mathrm{model}(\mathbf{x})$$
- is a *probability* vector
    - Given possible labels/classes
    $$C = \{c_1, \ldots, c_{\#\mathrm{C}}\}$$
    - y_j is the probability that input $\mathbf{x}$ is from class c_j

However: the HuggingFace standard is that $\mathbf{y}$ are **logits** rather than probabilities

- values *before* applying a softmax

The import of the difference is that

- the *loss function* must know
- that the model is returning *logits*, rather than *probabilities* (the Keras default)

In Keras, we can pass the loss either

- as a function object
    - e.g., `tf.keras.losses.SparseCategoricalCrossentropy`
- or a *string* denoting the function
    - e.g., `sparse_categorical_crossentropy`

To conform to the HuggingFace standard

- we should specify the loss as a function
- passing in an (optional) argument indicating that the model output are logits
    - e.g., `SparseCategoricalCrossentropy(from_logits=True)`

So the typical compile statement should look like

```
model.compile(
        ...,
        loss=SparseCategoricalCrossentropy(from_logits=True),
        ...)
```

rather than

```
model.compile(
        ...,
        loss='sparse_categorical_crossentropy',
        ...)
```

See the [warning for common pitfall (https://huggingface.co/learn/nlp-course/chapter3/3?fw=tf)](https://huggingface.co/learn/nlp-course/chapter3/3?fw=tf)

```
Note a very common pitfall here — you can just pass the name of the loss as a s
tring to Keras, but by default Keras will assume that you have already applied
a softmax to your outputs. Many models, however, output the values right before
the softmax is applied, which are also known as the logits. We need to tell the
loss function that that’s what our model does, and the only way to do that is t
o call it directly, rather than by name with a string.
```

In [ ]:

```
print("Done")
```