In [1]:
```
%run Latex_macros.ipynb
```

Ken Perry attribution:

- Derived from HuggingFace Course, Chapt 2, "Putting it all together" (https://huggin
  - Colab (https://colab.research.google.com/github/huggingface/notebooks/blob/m
    - link to Github repo no longer works
    - repo probably updated

Putting it all together (TensorFlow)

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

In [2]:
```
!pip install datasets evaluate transformers[sentencepiece]
```

```
Requirement already satisfied: datasets in /usr/local/lib/python3.10/dist-pack
ages (2.18.0)
Requirement already satisfied: evaluate in /usr/local/lib/python3.10/dist-pack
ages (0.4.1)
Requirement already satisfied: transformers[sentencepiece] in /usr/local/lib/p
ython3.10/dist-packages (4.38.2)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-pack
ages (from datasets) (3.13.3)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/dist-p
ackages (from datasets) (1.25.2)
Requirement already satisfied: pyarrow>=12.0.0 in /usr/local/lib/python3.10/di
st-packages (from datasets) (14.0.2)
Requirement already satisfied: pyarrow-hotfix in /usr/local/lib/python3.10/dis
t-packages (from datasets) (0.6)
Requirement already satisfied: dill<0.3.9,>=0.3.0 in /usr/local/lib/python3.1
0/dist-packages (from datasets) (0.3.8)
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packag
es (from datasets) (2.0.3)
Requirement already satisfied: requests>=2.19.0 in /usr/local/lib/python3.10/d
ist-packages (from datasets) (2.31.0)
Requirement already satisfied: tqdm>=4.62.1 in /usr/local/lib/python3.10/dist-
packages (from datasets) (4.66.2)
Requirement already satisfied: xxhash in /usr/local/lib/python3.10/dist-packag
es (from datasets) (3.4.1)
Requirement already satisfied: multiprocess in /usr/local/lib/python3.10/dist-
packages (from datasets) (0.70.16)
Requirement already satisfied: fsspec[http]<=2024.2.0,>=2023.1.0 in /usr/loca
l/lib/python3.10/dist-packages (from datasets) (2023.6.0)
Requirement already satisfied: aiohttp in /usr/local/lib/python3.10/dist-packa
ges (from datasets) (3.9.3)
Requirement already satisfied: huggingface-hub>=0.19.4 in /usr/local/lib/pytho
n3.10/dist-packages (from datasets) (0.20.3)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-pac
kages (from datasets) (24.0)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.10/dist-p
ackages (from datasets) (6.0.1)
Requirement already satisfied: responses<0.19 in /usr/local/lib/python3.10/dis
t-packages (from evaluate) (0.18.0)
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.10/
dist-packages (from transformers[sentencepiece]) (2023.12.25)
Requirement already satisfied: tokenizers<0.19,>=0.14 in /usr/local/lib/python
3.10/dist-packages (from transformers[sentencepiece]) (0.15.2)
Requirement already satisfied: safetensors>=0.4.1 in /usr/local/lib/python3.1
0/dist-packages (from transformers[sentencepiece]) (0.4.2)
Requirement already satisfied: sentencepiece!=0.1.92,>=0.1.91 in /usr/local/li
b/python3.10/dist-packages (from transformers[sentencepiece]) (0.1.99)
Requirement already satisfied: protobuf in /usr/local/lib/python3.10/dist-pack
ages (from transformers[sentencepiece]) (3.20.3)
Requirement already satisfied: aiosignal>=1.1.2 in /usr/local/lib/python3.10/d
ist-packages (from aiohttp->datasets) (1.3.1)
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.10/dist
-packages (from aiohttp->datasets) (23.2.0)
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.10/
```

```
dist-packages (from aiohttp->datasets) (1.4.1)
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3.1
0/dist-packages (from aiohttp->datasets) (6.0.5)
Requirement already satisfied: yarl<2.0,>=1.0 in /usr/local/lib/python3.10/dis
t-packages (from aiohttp->datasets) (1.9.4)
Requirement already satisfied: async-timeout<5.0,>=4.0 in /usr/local/lib/pytho
n3.10/dist-packages (from aiohttp->datasets) (4.0.3)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/py
thon3.10/dist-packages (from huggingface-hub>=0.19.4->datasets) (4.10.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/pyth
on3.10/dist-packages (from requests>=2.19.0->datasets) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-
packages (from requests>=2.19.0->datasets) (3.6)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.1
0/dist-packages (from requests>=2.19.0->datasets) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.1
0/dist-packages (from requests>=2.19.0->datasets) (2024.2.2)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python
3.10/dist-packages (from pandas->datasets) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-
packages (from pandas->datasets) (2023.4)
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/dis
t-packages (from pandas->datasets) (2024.1)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-pack
ages (from python-dateutil>=2.8.2->pandas->datasets) (1.16.0)
```

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 [3]:
```
from transformers import AutoTokenizer

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

```
/usr/local/lib/python3.10/dist-packages/huggingface_hub/utils/_token.py:88: Us
erWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab
(https://huggingface.co/settings/tokens), set it as secret in your Google Cola
b and restart your session.
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access pu
blic models or datasets.
  warnings.warn(
```

Let's understand the Tokenizer

In [4]:
```
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 [5]:
```
print("Text: ", sequence)
```

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

In [6]:
```
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 [7]:
```
# 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 [8]:
```
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 [9]:
```
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 PyTorch model weights were used when initializing TFDistilBertForSequenceC
lassification.

All the weights of TFDistilBertForSequenceClassification were initialized from
the PyTorch model.
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 [10]:
```
output
```

Out[10]:
```
TFSequenceClassifierOutput(loss=None, logits=<tf.Tensor: shape=(2, 2), dtype=f
loat32, numpy=
array([[-1.560696 ,  1.6122813],
       [-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 [11]:
```
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
$$\mathbf{y} = \text{model}(\mathbf{x})$$
- is a *probability* vector
  - Given possible labels/classes
$$C = \{c_1, \ldots, c_{\#\text{C}}\}$$
  - 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
```

Remember

- the Loss function must be compatible with the type of the model output
    - logits or probabilties

Examining the model

- inspect the `__init__` and `call` methods

`__init__` will show the model components

- we can recursively inspect the components

`call` will show you how the model transforms input to output

In [12]:
```
model.__init__??
```

We will recursively examine the `distilbert` attribute which is a `TFDistilBertMainLayer`

But first, let's examine the `call` method, which will help us understand how the components are connected.

In [13]:
```
model.call??
```

If you scroll through all the description, you will find the body of the call

- the `distilbert` component is called, result assigned to `distilbert_output`
- the `distilbert_output` is passed through several layers

Let's work our way down the model, starting with the `distilbert` attribute is what we want

In [14]:
```
model.distilbert
```

Out[14]:
```
<transformers.models.distilbert.modeling_tf_distilbert.TFDistilBertMainLayer a
t 0x7e4979a15e40>
```

We can see that it is of type `TFDistilBertMainLayer`

Use the Jupyter notebook ? and ?? tools to inspect

- the signature
- the code

In [15]:
```
model.distilbert?
```