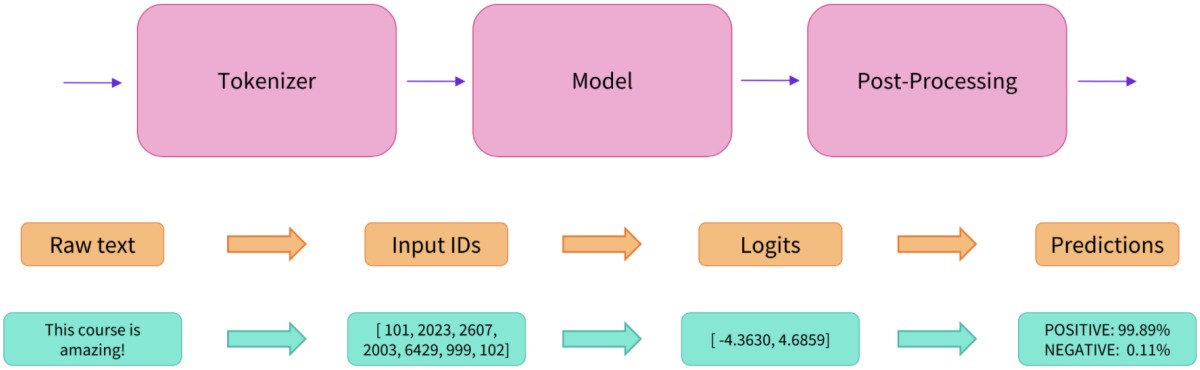
Using 🤗 Transformers

M2 I1. Behind the pipeline



When you use a pretrained Transformer for your down stream task, we have to make sure that the tokenizer used is same as the one which was used while building the pretrained model.

To do this, we can use the `AutoTokenizer` class and its `from_pretrained()` method. Using the checkpoint name of our model, it will automatically fetch the data associated with the model's tokenizer and cache it

```
from transformers import AutoTokenizer

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

# PreTrainedTokenizerFast(
# name_or_path='distilbert-base-uncased-finetuned-sst-2-english',
# vocab_size=30522, model_max_len=512, is_fast=True, padding_side='right',
```

```
# special_tokens={'unk_token': '[UNK]', 'sep_token': '[SEP]',
# 'pad_token': '[PAD]', 'cls_token': '[CLS]', 'mask_token': '[MASK]'})
```

The tokenizer now can be passed sentences and we get back a dictionary of list of ID's which need to be converted to tensors.

Example:

```
raw_input = [
    'Hey I am for a test',
    'All the best'
]
inputs = tokenizer(raw_input, padding=True, truncation=True,
                   return_tensors='tf'
                  )
print(inputs)
# {'input_ids': <tf.Tensor: shape=(2, 8), dtype=int32, numpy=
# array([[ 101, 4931, 1045, 2572, 2005, 1037, 3231,  102],
#        [ 101, 2035, 1996, 2190,  102,    0,    0,    0]], dtype=int32)>,
# 'attention_mask': <tf.Tensor: shape=(2, 8), dtype=int32, numpy=
# array([[1, 1, 1, 1, 1, 1, 1, 1],
#        [1, 1, 1, 1, 1, 0, 0, 0]], dtype=int32)>}
```

attention_mask will be explanined later.

Going through the model

```
model = TFAutoModel.from_pretrained(checkpoint)
```

This architecure contains only the base Transformer module, given some inputs it will output **hidden states/features** . For each model input, we'll retrieve a high-dimensional vector representing the **contextual understanding of that input by the Transformer model**.

The output high-dimensional Vector

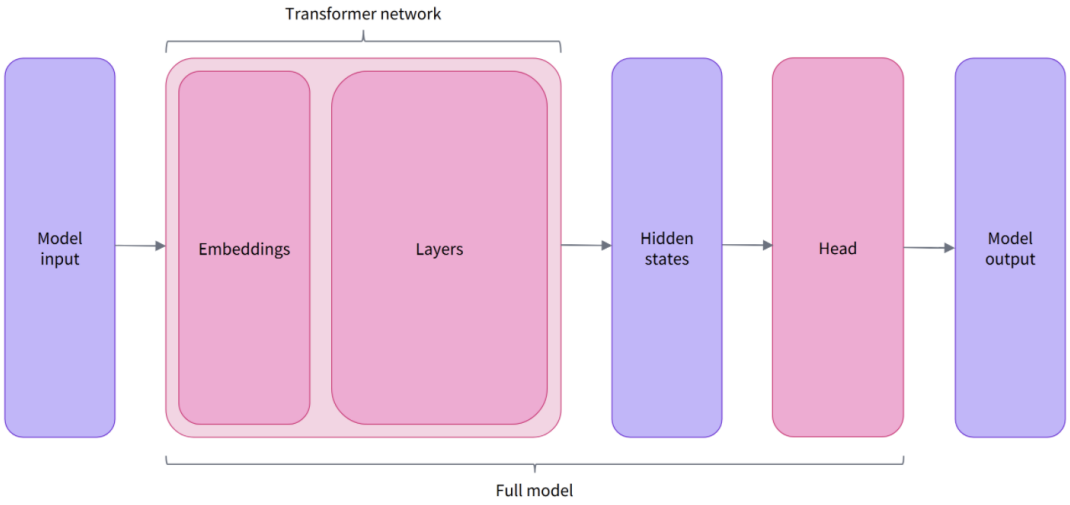
```
outputs = model(inputs)
print(outputs.last_hidden_state.shape)

#(2, 8, 768)
```

These vectors so returned are the input to the head of the Transformer.

The vector output by the Transformer module is usually large. It generally has three dimensions:

- **Batch size**: The number of sequences processed at a time (2 in our example).
- **Sequence length**: The length of the numerical representation of the sequence (8 in our example).
- **Hidden size**: The vector dimension of each model input.

Model heads



The output of the Transformer model is sent directly to the model head to be processed.

1. Our input sentence is broken down into and converted to input id’s
2. Then these input id’s are mapped to vectors (through dictonary look ups)

3. The subsequent layers manipulate those vectors using the attention mechanism to produce the final representation of the sentences.

For a text classification task like we have we won't be using the TFAutoModel Class but we will use the TFAutoModelForSequenceClassification.

```
senti_model = TFAutoModelForSequenceClassification.from_pretrained(checkpoint)
outputs = senti_model(inputs)

# TFSequenceClassifierOutput(loss=None, logits=<tf.Tensor: shape=(2, 2),
# dtype=float32, numpy=
# array([[-1.4058973,  1.5513633],
#        [-4.28962  ,  4.6092067]], dtype=float32)>,
# hidden_states=None, attentions=None)
```

The output array so returned i.e :

For sentence 1 from our raw_input we get [-1.4058973, 1.5513633]

For sentence 2 from our raw_input we get [-4.28962 ， 4.6092067]

These are logits and not the output probablities. To get the probablity score we have pass these logits through a softmax. We will use tensorflow for getting our probablity scores.

Postprocessing the output

```
print(outputs.logits)
# tf.Tensor(
# [[-1.4058973  1.5513633]
# [-4.28962    4.6092067]], shape=(2, 2), dtype=float32)
```

Now lets get our probablities:

```
import tensorflow as tf
predictions = tf.math.softmax(outputs.logits, axis=-1)
print(predictions)

# tf.Tensor(
# [[4.9394473e-02 9.5060551e-01]
# [1.3653041e-04 9.9986351e-01]], shape=(2, 2), dtype=float32)
```

The sentence 1 has probablites: [0.049, 0.950]

The sentence 2 has probablites: [0.0001, 0.998]

We can check the labels for the task using. the following code

```
model.config.id2label
# {0: 'NEGATIVE', 1: 'POSITIVE'}
```

Therefore our results are as follows:

First Sentence NEGATIVE: 0.049, POSITIVE: 0.950

Second Sentence NEGATIVE: 0.0001, POSITIVE: 0.998