In today's lab!

- Learn what is behind the pipeline
  - pre-processing
    - tokenization
  - passing the input through the model
  - post-processing
    - convert model output into meaningfull information

1/29

1

Recap pipeline()
-> opening chap2.2 notebook in colab

- Open HuggingFace course chapter 2.2
  - open in colab

```
from transformers import pipeline

classifier = pipeline("sentiment-analysis")
classifier(
    [
        "I've been waiting for a HuggingFace course my whole life.",
        "I hate this so much!",
    ]
)
```

Name of the default model

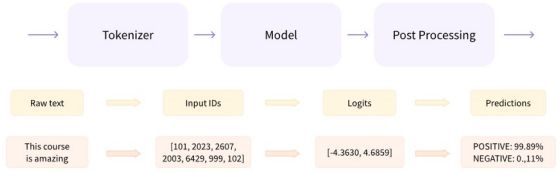
```
No model was supplied, defaulted to distilbert-base-uncased-finetuned-sst-2-english (https://huggingface.co/distilbert-base-uncased-finetuned-sst-2-english)
Downloading: 100%  629/629 [00:00<00:00, 10.2kB/s]
Downloading: 100%  255M/255M [00:07<00:00, 31.9MB/s]
Downloading: 100%  48.0/48.0 [00:00<00:00, 1.11kB/s]
Downloading: 100%  226k/226k [00:00<00:00, 868kB/s]
[{'label': 'POSITIVE', 'score': 0.9598048329353333},
 {'label': 'NEGATIVE', 'score': 0.9994558691978455}]
```

2/29

2

Behind pipeline()

- 3 steps
  - pre-processing
  - passing inputs through the model
  - post-processing



3/29

3

Pre-processing underlying pipeline()

- Neural networks cannot process raw text
  - split text into tokens
  - map each token to an integer
    - words -> problems with unknown words
    - characters -> eventually too long sequences
    - sub-words -> halfway solution
  - map each integer into an embbeding
- Some sub-word algorithms
  - Byte Pair Encoding (BPE)
  - WordPiece
  - Unigram
  - SentencePiece
- Lets see an example with Byte Pair Encoding (BPE)…

4/29

4

Byte-Pair Encoding (BPE)

- Pre-tokenization step
  - split text into words
  - word frequency count

```
("hug", 10), ("pug", 5), ("pun", 12), ("bun", 4), ("hugs", 5)
```

- Create a vocabulary with all the symbols in the corpus

```
["b", "g", "h", "n", "p", "s", "u"]
("h" "u" "g", 10), ("p" "u" "g", 5), ("p" "u" "n", 12), ("b" "u" "n", 4), ("h" "u" "g" "s", 5)
```

- Create new vocabulary entries by most frequent pair

5/22

5

Byte-Pair Encoding (BPE)

```
["b", "g", "h", "n", "p", "s", "u"]
("h" "u" "g", 10), ("p" "u" "g", 5), ("p" "u" "n", 12), ("b" "u" "n", 4), ("h" "u" "g" "s", 5)
```

- ug -> 10+5+5 = 20

```
("h" "ug", 10), ("p" "ug", 5), ("p" "u" "n", 12), ("b" "u" "n", 4), ("h" "ug" "s", 5)
```

- un -> 12+4 = 16
- hug -> 10+5 = 15

```
["b", "g", "h", "n", "p", "s", "u", "ug", "un", "hug"]
("hug", 10), ("p" "ug", 5), ("p" "un", 12), ("b" "un", 4), ("hug" "s", 5)
```

6/22

6

Byte-Pair Encoding (BPE)

- We stop when the desired vocabulary size is reached
  - vocabularies of sizes from 16000 to 48000 are common

```
["b", "g", "h", "n", "p", "s", "u", "ug", "un", "hug"]
```

- Bug -> [“b”, “ug”]
- Mug -> [“<unk>, “ug”]

- <unk> is very unlikely as most symbols are present in the initial corpus
  - might occur with emojis or characters from different languages (Chinese, Japonese…)

- Other sub-word algorithms are similar

7/29

7

AutoTokenizer class ---- supporting preprocessing

- Neural networks cannot process raw text
  - split text into tokens
    - words, sub-words, characters
  - map each token to an integer and then from integer to embedding
- Tokenizer
  - AutoTokenizer() class -> to be imported like pipeline was imported
  - from_Pretrained() method -> “function” that loads a pre-trained tokenizer

Similar to pipeline

```
from transformers import AutoTokenizer

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

Name or path of the model (with its tokenizer)

8/29

8

from_pretrained() method

```
raw_inputs = [
    "I've been waiting for a HuggingFace course my whole life.",
    "I hate this so much!",
]
inputs = tokenizer(raw_inputs, padding=True, truncation=True, return_tensors="pt")
print(inputs)
```

- tokenizer() -> transforms sequences into tokens and tokens into integers
  - outputs a dictionary with the inputs to the model

9/29

9

Tensor output by tokenization

```
{
    'input_ids': tensor([
        [  101,  1045,  1005,  2310,  2042,  3403,  2005,  1037, 17662, 12172, 2607,  2026,  2878,  2166,  1012,   102],
        [  101,  1045,  5223,  2023,  2061,  2172,   999,   102,     0,     0,     0,     0,     0,     0,     0,     0]
    ]),
    'attention_mask': tensor([
        [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
        [1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
    ])
}
```

- Dictionary has two entries ‘input_ids’ and ‘attention_mask’ that contain tensors
  - attention_mask -> informs the model where padding was added
- Tensor is a data structure of the pytorch library that NN models can handle

10/29

10

parameters for from_pretrained() method

```
inputs = tokenizer(raw_inputs, padding=True, truncation=True, return_tensors="pt")
```

- tokenizer()
  - padding -> “filler” so every sentence has the same length
  - truncation -> shorten sentences to limit input size
  - return_tensors -> return sequences in “DL library format” (pt=pytorch, tf=tensorflow)

```
{
    'input_ids': tensor([
        [  101,  1045,  1005,  2310,  2042,  3403,  2005,  1037, 17662, 12172, 2607,  2026,  2878,  2166,  1012,   102],
        [  101,  1045,  5223,  2023,  2061,  2172,   999,   102,     0,     0,     0,     0,     0,     0,     0,     0]
    ]),
    'attention_mask': tensor([
        [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
        [1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
    ])
}
```

11/29

11

Playing with parameters for tokenization
-> “enriching” to the chap2.2 notebook in colab

```
raw_inputs = [
    "I've been waiting for a HuggingFace course my whole life.",
    "I hate this so much!",
]
inputs = tokenizer(raw_inputs, padding=True, truncation=True, return_tensors="pt")
print(inputs)
```

- Ex. 1-
  a. Pass padding=“max_length” to the tokenizer, what happens?
  b. Pass a new argument max_length=20, what happens?
  c. Pass padding=“longest” to the tokenizer, what happens?
  d. Pass a new argument max_length=10, what happens?
  e. Pass truncation=False, what happens?

12/29

12

substeps in tokenization: get tokens

```
sentence = "I've been waiting for a HuggingFace course my whole life."
tokens = tokenizer.tokenize(setence)
print(tokens)
```

- tokenizer.tokenize()
  - tokenizes text without converting tokens into integers

```
['i', "'", 've', 'been', 'waiting', 'for', 'a', 'hugging', '##face', 'course', 'my', 'whole', 'life', '.']
```

- Ex. 2 -
  a. Change the text to another language. (e.g. “Estive à espera…)
  b. Try a website, a phone number, a math equation.
  c. Try gibberish.

13/29

13

substeps in tokenization: get ids

- tokenizer.convert_tokens_to_ids()
  - maps the tokens to integers

```
ids = tokenizer.convert_tokens_to_ids(tokens)
print(ids)
```

```
[1045, 1005, 2310, 2042, 3403, 2005, 1037, 17662, 12172, 2607, 2026, 2878, 2166, 1012]
```

14/29

14

substeps in tokenization: back to raw input

- tokenizer.decode()
  - maps the integers to tokens

```
decoded_string = tokenizer.decode(ids)
print(decoded_string)
```

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

- Ex. 3 –
  a. Repeat exercise 1 with convert_tokens_to_ids() and decode().

15/29

15

```
sentence = "I've been waiting for a HuggingFace course my whole life."
tokens = tokenizer.tokenize(sentence)
ids = tokenizer.convert_tokens_to_ids(tokens)
decoded_string = tokenizer.decode(ids)
print(decoded_string)
```

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

16/29

16

AutoModel class ---- supporting ~~pre~~processing
-> resuming to the chap2.2 notebook in colab

- AutoModel class -> to be imported like pipeline was imported
  - loads model from name/path

```
from transformers import AutoModel

checkpoint = "distilbert-base-uncased-finetuned-sst-2-english"
model = AutoModel.from_pretrained(checkpoint)
outputs = model(**inputs)
```

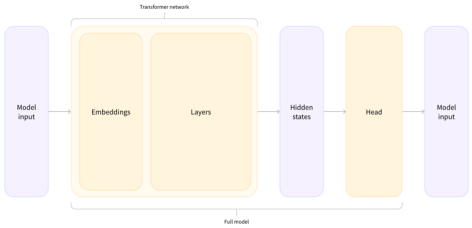
"transforms" inputs into arguments

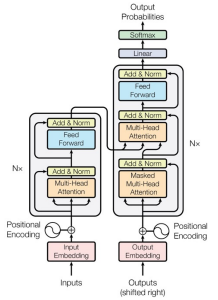
- Outputs contains the last layer's hidden state
  - still needs to pass through a head layer to perform a specific task

17/22

17

The model/processing inside pipeline()





18/29

18

last_hidden_state

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

```
torch.Size([2, 16, 768])
```

- batch size/number of input sequences
- model hidden size
- sequence length

- Ex. 4 –
  a. Print the contents of outputs.
  b. Add a sentence to raw_inputs and re-run the relevant code. Print the shape.
  c. Add max_length = 8 to tokenizer and re-run relevant code. Print the shape.

19/29

19

The model in pipeline()

- AutoModel (retrieve the hidden states)
- AutoModelForCausalLM
- AutoModelForMaskedLM
- AutoModelForMultipleChoice
- AutoModelForQuestionAnswering
- AutoModelForSequenceClassification
- AutoModelForTokenClassification
- …

- More on this in a few classes

20/29

20

AutoModelForSequenceClassification class

- The sentiment analysis task is a classification task

```
from transformers import AutoModelForSequenceClassification

checkpoint = "distilbert-base-uncased-finetuned-sst-2-english"
model = AutoModelForSequenceClassification.from_pretrained(checkpoint)
outputs = model(**inputs)
```

21/29

21

logits

```
print(outputs.logits.shape)
```

```
torch.Size([2, 2])
```

→ 2 sequences and 2 possible outputs

```
print(outputs.logits)
```

```
tensor([[-1.5607,  1.6123],
        [ 4.1692, -3.3464]], grad_fn=<AddmmBackward>)
```

→ output still needs to pass trough a softmax

22/29

22

AutoModelForSequenceClassification class ---- supporting postprocessing

- Softmax
  - function that transforms input into a series of probabilities
    - between 1 and 0
    - sum of all probabilities equals 1

```
import torch

predictions = torch.nn.functional.softmax(outputs.logits, dim=-1)
print(predictions)
```

```
tensor([[4.0195e-02, 9.5980e-01],
        [9.9946e-01, 5.4418e-04]], grad_fn=<SoftmaxBackward>)
```

→ sum equals 1

```
model.config.id2label
```

```
{0: 'NEGATIVE', 1: 'POSITIVE'}
```

23/29

23

AutoModelForSequenceClassification class ---- supporting postprocessing

- softmax()

```
import torch

predictions = torch.nn.functional.softmax(outputs.logits, dim=-1)
print(predictions)
```

```
tensor([[4.0195e-02, 9.5980e-01],
        [9.9946e-01, 5.4418e-04]], grad_fn=<SoftmaxBackward>)
```

- id2label

```
model.config.id2label
```

```
{0: 'NEGATIVE', 1: 'POSITIVE'}
```

24/29

24

Wrapping Up
-> moving to chap2.5 notebook in colab

- Open HuggingFace course chapter 2.5
  - open in colab
- Putting the pipeline together

25/29

25

Wrapping Up

```
import torch
from transformers import AutoTokenizer, AutoModelForSequenceClassification

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

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

tokens = tokenizer.tokenize(sequence)
ids = tokenizer.convert_tokens_to_ids(tokens)
input_ids = torch.tensor(ids)
# This line will fail.
model(input_ids)
```

Creates a list of tokens

Transforms list into tensor

```
IndexError: Dimension out of range (expected to be in range of [-1, 0], but got 1)
```

By default the model expects multiple sentences, i.e. a list of lists of tokens.

26/29

26

Wrapping Up

- Only need to pass ids inside a list

```
input_ids = torch.tensor([ids])
print("Input IDs:", input_ids)

output = model(input_ids)
print("Logits:", output.logits)
```

```
Input IDs: tensor([[ 1045,  1005,  2310,  2042,  3403,  2005,  1037, 17662, 12172,  2607,
          2026,  2878,  2166,  1012]])
Logits: tensor([[-2.7276,  2.8789]], grad_fn=<AddmmBackward0>)
```

27/29

27

Wrapping Up

```
model = AutoModelForSequenceClassification.from_pretrained(checkpoint)

sequence1_ids = [[200, 200, 200]]
sequence2_ids = [[200, 200]]
batched_ids = [
    [200, 200, 200],
    [200, 200, tokenizer.pad_token_id],
]

print(model(torch.tensor(sequence1_ids)).logits)
print(model(torch.tensor(sequence2_ids)).logits)
print(model(torch.tensor(batched_ids)).logits)
```

```
tensor([[ 1.5694, -1.3895]], grad_fn=<AddmmBackward0>)
tensor([[ 0.5803, -0.4125]], grad_fn=<AddmmBackward0>)
tensor([[ 1.5694, -1.3895],
        [ 1.3374, -1.2163]], grad_fn=<AddmmBackward0>)
```

Should be the same. We need to pass the attention_mask

28/29

28

Wrapping Up

```
batched_ids = [
    [200, 200, 200],
    [200, 200, tokenizer.pad_token_id],
]

attention_mask = [
    [1, 1, 1],
    [1, 1, 0],
]

outputs = model(torch.tensor(batched_ids), attention_mask=torch.tensor(attention_mask))
print(outputs.logits)
```

- Same result as passing the sequences individually

```
tensor([[ 1.5694, -1.3895],
        [ 0.5803, -0.4125]], grad_fn=<AddmmBackward0>)
```

29/29

29

Wrapping Up

- Ex. 5 –
  a. Build the whole pipeline for the sentences:
     i. "*I've been waiting for a HuggingFace course my whole life.*"
     ii. "*I hate this so much!*"
     iii. Pass them together to the model. **Hint:** print the ids of each sequence and pass them to the model by using batched_ids and attention_mask as in the previous slide.
     iv. Use tokenizer(raw_inputs) directly.

30/29

30