Having fun with `wav2vec` (final version)

Warning it is strongly advised to do this lab in google colab (in spite of all the "ethical" problems resulting from using this service).

`wav2vec` is a neural network using a `transformer` architecture to model speech and `xlsr-53` is a pre-trained multilingual model of `wav2vec` : using this model, it is possible to build a vector representation of an audio segment in any language that can be used for any speech processing task. In this lab, we will focus on the *automatic transcription* task whose objective is to convert an audio input into text and we will use these representation to evaluate the similarity between languages.

We will start by installing the different libraries that we will use:

In [ ]:
```
!pip install ffmpeg-python
!pip install transformers
!pip install evaluate
!pip install jiwer
!apt-get install ffmpeg
!apt-get install libavutil-dev
```

Questions

1. Why can't we use `pip` to install `ffmpeg`?

Recording wav files using the computer microphone

Remarque : If you don't have access to a computer with a microphone, you can skip to the next part (and come back to this one later)

The next cell defines a function `get_audio()` which allows you to record an audio file from the computer's microphone. This function can only be run in a browser (i.e. using google colab). It returns two variables:

- the audio data
- the samping rate

This data can be stored into a wav file using the following instructions:

```
import scipy

audio, sr = get_audio()
scipy.io.wavfile.write('my_file.wav', sr, audio)
```

Do not forget that this file is stored in a virtual machine and will be deleted as soon as this machine is shutdown. You can download it to your computer with:

```
from google.colab import files
files.download("/content/mon_fichier.wav")
```

In [ ]:

```
"""
code of Noé Tits - Numediart (UMONS) - noetits.com
https://colab.research.google.com/drive/1Z6VIRZ_sX314hyev3Gm5gBqvm1wQVo-a#scrollTo=uNzFb
"""
from IPython.display import HTML, Audio
from google.colab.output import eval_js
from base64 import b64decode
import numpy as np
from scipy.io.wavfile import read as wav_read
import io
import ffmpeg

AUDIO_HTML = """
<script>
var my_div = document.createElement("DIV");
var my_p = document.createElement("P");
var my_btn = document.createElement("BUTTON");
var t = document.createTextNode("Press to start recording");

my_btn.appendChild(t);
//my_p.appendChild(my_btn);
my_div.appendChild(my_btn);
document.body.appendChild(my_div);

var base64data = 0;
var reader;
var recorder, gumStream;
var recordButton = my_btn;

var handleSuccess = function(stream) {
  gumStream = stream;
  var options = {
    //bitsPerSecond: 8000, //chrome seems to ignore, always 48k
    mimeType : 'audio/webm;codecs=opus'
    //mimeType : 'audio/webm;codecs=pcm'
  };
  //recorder = new MediaRecorder(stream, options);
  recorder = new MediaRecorder(stream);
  recorder.ondataavailable = function(e) {
    var url = URL.createObjectURL(e.data);
    var preview = document.createElement('audio');
    preview.controls = true;
    preview.src = url;
    document.body.appendChild(preview);

    reader = new FileReader();
    reader.readAsDataURL(e.data);
    reader.onloadend = function() {
      base64data = reader.result;
      //console.log("Inside FileReader:" + base64data);
    }
  };
  recorder.start();
  };

recordButton.innerText = "Recording... press to stop";

navigator.mediaDevices.getUserMedia({audio: true}).then(handleSuccess);


function toggleRecording() {
  if (recorder && recorder.state == "recording") {
      recorder.stop();
      gumStream.getAudioTracks()[0].stop();
      recordButton.innerText = "Saving the recording... pls wait!"
  }
```

```
}

// https://stackoverflow.com/a/951057
function sleep(ms) {
  return new Promise(resolve => setTimeout(resolve, ms));
}

var data = new Promise(resolve=>{
//recordButton.addEventListener("click", toggleRecording);
recordButton.onclick = ()=>{
toggleRecording()

sleep(2000).then(() => {
  // wait 2000ms for the data to be available...
  // ideally this should use something like await...
  //console.log("Inside data:" + base64data)
  resolve(base64data.toString())

});

}
});

</script>
"""

def get_audio():
  display(HTML(AUDIO_HTML))
  data = eval_js("data")
  binary = b64decode(data.split(',')[1])

  process = (ffmpeg
    .input('pipe:0')
    .output('pipe:1', format='wav')
    .run_async(pipe_stdin=True, pipe_stdout=True, pipe_stderr=True, quiet=True, overwrit
  )
  output, err = process.communicate(input=binary)

  riff_chunk_size = len(output) - 8
  # Break up the chunk size into four bytes, held in b.
  q = riff_chunk_size
  b = []
  for i in range(4):
      q, r = divmod(q, 256)
      b.append(r)

  # Replace bytes 4:8 in proc.stdout with the actual size of the RIFF chunk.
  riff = output[:4] + bytes(b) + output[8:]

  sr, audio = wav_read(io.BytesIO(riff))

  return audio, sr
```

Question

2. Create an audio corpus consisting of about ten sentences (300 words) in the language of your choice (you should check that there exists a transcription model for the language you choose — see below) and their transcription. Each sentence should be stored in a different `wav` file. Transcription is a time-consuming task that is not very interesting (rule of thumb: it takes 10 minutes to transcribe 1 minute of audio). It is easier to read existing sentences than to transcribe new data from scratch. Make sure you choose sentences of different registers (e.g. sentences with technical terms, proper nouns or everyday language)

Using a pre-trained model

The `HuggingFace` 🤗 library provides a simple way to use pretrained models. It defines an API (set of functions) that allow you to:

- automatically download a model from its identifier (a string) and easily retrieve the models corresponding to a language/task thanks to the "model hub". The downloaded models are stored in a cache).
- provide high-level methods for manipulating and using these models

We will first use a pre-trained model to automatically recognize French recordings (`jonatasgrosman/wav2vec2-large-xlsr-53-french`). This model has been trained by fine-tuning the XLS-R model on a French corpus.

You will find pre-trained models for all languages (almost) on the model hub.

Questions

3. Why are downloaded models cached?
4. Are there other speech recognition models for French available on HuggingFace 🤗 model hub.

HuggingFace 🤗 uses `DataFrame` to store and manipulate annotated corpora. A `DataFrame` is a data structure defined in the `pandas` library which is today the main data analysis library in Python to represent tabular data (caricaturally, a sheet in a spreadsheet): each row represents an *example* which is described by a set of *variables* or *features* corresponding to the columns.

The following code allows to initialize the `DataFrame` corresponding to the corpus we are going to use:

In [ ]:
```
import pandas as pd

# corpus is a list of dictionaries
# there is one dictionary for each example
# the column names are the keys
corpus = [
    {"filename": "/content/sentence_01.wav",
     "transcription": "Bonjour les amis"},
    {"filename": "/content/sentence_02.wav",
     "transcription": "Comment allez-vous"}
]

corpus = pd.DataFrame(corpus)
corpus
```

Questions

5. Build a `DataFrame` to store the corpus collected in the previous question
6. Build a `DataFrame` to store the French corpus avalaible in the Moodle: the corpus is made of several audio files and a json file that maps the name of an audio file to its transcription

Loading audio files

The code in the next cell adds a column to the `DataFrame` we have juste created (`corpus`). To do this, the `apply` method will call the `load4xslr` function for each value in the `filename` column. This function, in addition to "loading" the file, will make sure that it is in a format suitable for the model (in particular that it is sampled at $16,000$ Hz).

For each row, the `audio` column will contain a row vector corresponding the the audio data: the number of elements in this matrix will be proportional to the length of the audio file and the i-th value (a real number) will correspond to the value of the audio signal at the i-th time step.

In [ ]:
```
import torchaudio
import librosa

from pathlib import Path

import numpy as np


def load4xslr(audio_filepath: str, window_size: int=None, frame_offset: int=None) -> np.
    """
    Load a wav file and ensure that it can be used in XSL-R/wav2vec

    To be used by XSL-R, wav files have to:
    - have a samping rate of 16_000Hz
    - use only one channel (mono file)

    Parameters:
    - audio_filepath, str --> path to the file
    - window_size, int, frame_offset --> if not None, return `window_size` ms of audio d
            centered around `frame_offset`; if both None, returns the full file.
    """
    speech_array, sampling_rate = librosa.load(Path(audio_filepath))
    speech = librosa.to_mono(speech_array)

    speech = librosa.resample(speech, orig_sr=sampling_rate, target_sr=16_000)

    if frame_offset is None and window_size is None:
        return speech

    if frame_offset is None:
        frame_offset = speech.shape[0] // 2

    assert speech.shape[0] / sampling_rate > window_size / 1_000,\
        f"File too short ({speech.shape[0] / sampling_rate:.2f}s) to extract a {window_s

    window_size = int(16_000 * window_size // 2_000)

    return speech[frame_offset - window_size:frame_offset + window_size]

corpus["audio"] = corpus["filename"].apply(load4xslr)
```

The following code allows you to access to the signal corresponding to the record of the line `n` (the attribute `loc` of a `DataFrame` allows to access a value by specifying its line and column).

In [ ]:
```
n = 10
print(f"size of the matrice: {corpus.loc[n, 'audio'].shape}")
print(f"corresponding values: {corpus.loc[n, 'audio']}")
```

The signal can also be visualized using the following function:

In [ ]:
```
import matplotlib.pyplot as plt

def plot_waveform(waveform, sample_rate):
    figure, axes = plt.subplots(1, 1)
    axes.plot(range(waveform.shape[0]), waveform, linewidth=1)
    plt.show(block=False)

plot_waveform(corpus.loc[n, "audio"], 16_000)
```

Questions

7. What will be the size of the matrix storing a 3 minute and 17 second audio signal?

Automatic transcription

The following cell will perform an automatic transcription of the corpus (stored in a new column called `automatic_transcription`) by:

1. downloading the pretrained modls (using the `from_pretrained` functions)
2. pre-processing the audio-data to ensure they are in the expected format (appel à `processor`)
3. transcribe the audio by "calling" the `model` to predict the probability that an audio frame is associated to a character and looking for the highest-scoring character. This probabilities are stored in a *tensor* (a 3 dimensions matrix).

In [ ]:
```
import torch

from transformers import Wav2Vec2ForCTC, Wav2Vec2Processor

model_id = "jonatasgrosman/wav2vec2-large-xlsr-53-french"

processor = Wav2Vec2Processor.from_pretrained(model_id)
model = Wav2Vec2ForCTC.from_pretrained(model_id)

inputs = processor(list(corpus["audio"]),
                   sampling_rate=16_000,
                   return_tensors="pt",
                   padding=True)

with torch.no_grad():
    logits = model(inputs.input_values, attention_mask=inputs.attention_mask).logits

predicted_ids = torch.argmax(logits, dim=-1)
predicted_sentences = processor.batch_decode(predicted_ids)

corpus["automatic_transcription"] = pd.Series(predicted_sentences)
```

Questions

8. What is the shape (dimensions) of `logit`. What does each dimension represent?
9. Explain the instruction `torch.argmax(logits, dim=-1)`
10. Evaluate qualitatively the quality of the automatic transcription for the two corpora considered (the one you collected and the one on the moodle)

11. The quality of automatic transcriptions is generally evaluated using the *Word Error Rate* (WER). Describe intuitively how this metric is working

The HuggingFace 🤗 library contains functions for almost all existing metrics, including one to calculate the WER:

```
from evaluate import load
wer = load("wer")
wer_score = wer.compute(predictions=...,
                        references=...)
```

This function takes as parameters, two lists of sentences (or more precisely, two *iterable*s i.e. python structures that can be used in a `for` loop).

Questions

12. Compute the WER on your two corpora?
13. Compute the WER between `Le chat, le chien et le lamantin` and `le chat le chien et le lamantin`. What can you conclude about the validity of the score you just calculated?

It is possible to apply string methods (e.g. `upper`) to all elements of a column with:

```
df["nouvelle_colonne"] = df["col"].str.upper()
```

Questions

14. Compute the WER after normalizing the references and the automatic transcription (to make them more “comparable”). What can you conclude?

Language distance

We will now use the representations computed by `XSL-R` to measure the similarity between different languages.

You will find, on the moodle, an archive (`commonvoice_sample.tar`) containing audio recording in different European (or not)languages. The language of a file can be extracted from the filename.

Questions

14. Write a function that takes as parameters a language, a window size and a number of segments `n` and returns a list of `n` segments of `window_size` seconds in a given language sampled randomly from all files.
15. Write a function that takes as parameter a list of audio segments and uses `XSL-R` to compute a list containing a representation of each segment (using the function in the following cell).
16. Given two languages compute the *distance* between them as the mean cosine distance between all segments of the two languages. We will consider 100 segments of 5s. You can/should use the `pairwise_distances` function of the from `sklearn.metrics` package.

17. How is the cosine distance defined? Why do we consider this distance rather than, for instance, the Euclidian distance?
18. Compute the distance between all languages in the corpus (including the language itself) and store the result as list of dictionnaries `[{"lang1": ..., "lang2": ..., "distance": ...}, {"lang1": ..., "lang2": ..., "distance": ...}, ...`

In [ ]:
```python
from typing import Tuple

def load_xlsr53_model(device: str = "cpu") -> Tuple[Wav2Vec2ForCTC, Wav2Vec2Processor]:
    """
    Load the XLSR-53 Large model (i.e. processor and model).

    As the model is not fine-tuned, we need to manually build the processor
    """

    import tempfile
    import json
    from transformers import Wav2Vec2FeatureExtractor
    from transformers import Wav2Vec2CTCTokenizer

    ofile = tempfile.NamedTemporaryFile("wt")
    json.dump({"[UNK]": "0", "[PAD]": "1"}, ofile)
    ofile.flush()

    tokenizer = Wav2Vec2CTCTokenizer(ofile.name,
                                     unk_token="[UNK]",
                                     pad_token="[PAD]",
                                     word_delimiter_token="|")
    feature_extractor = Wav2Vec2FeatureExtractor(feature_size=1,
                                                 sampling_rate=16_000,
                                                 padding_value=0.0,
                                                 do_normalize=True,
                                                 return_attention_mask=True)
    xlsr_processor = Wav2Vec2Processor(feature_extractor=feature_extractor,
                                       tokenizer=tokenizer)
    xlsr_model = Wav2Vec2ForCTC.from_pretrained("facebook/wav2vec2-large-xlsr-53").to(de

    return xlsr_model, xlsr_processor

def get_xlsr_representation(
    speech_signal: np.ndarray,
    processor: Wav2Vec2Processor,
    model: Wav2Vec2ForCTC,
    device: str = "cpu",
    pooling: str = "max",
) -> np.ndarray:
    """
    Parameters
    ----------
    - device: on which device the audio signal should be loaded (either "cpu" or "cuda")
    - processor, model: the wav2vec model
    - pooling: wav2vec splits each second of the input signal into 49,000 segments and b
               a representation for each of these segments. The pooling strategy defines
               representations of these segments will be aggregated to build a single ve
               representing the whole audio signal. Possible values are: "max" and "mean
    """
    inputs = processor(speech_signal, sampling_rate=16_000, return_tensors="pt")

    inputs = {k: v.to(device) for k, v in inputs.items()}
    with torch.no_grad():
        hidden_state = model(**inputs, output_hidden_states=True).hidden_states
```

```
        # hidden_state is a list of n+1 tensors: one for the output of the embeddings an
        # for the output of each layer — here we we want the embeddings.
        embeddings = hidden_state[0]

        # embeddings is tensor of shape [batch_size, sequence_length, repr_size]
        #
        # batch size is always 1 — we are considering a single audio segment.
        # the encoder outputs representation at 49Hz --> sequence length is equal to 49

        if pooling == "max":
            speech_representation = embeddings[0].max(axis=0).values
        elif pooling == "mean":
            speech_representation = embeddings[0].mean(axis=0)
        else:
            raise Exception

    return speech_representation.cpu().numpy()
```

It is possible to cluster the distances you calculated hierarchically with the following two instructions:

In [ ]:
```
import pandas as pd
import seaborn as sns

df = pd.DataFrame(distances)
sns.clustermap(df.pivot(index="lang1", columns="lang2", values="distance"))
```

Questions

18. What is the hierarchical clustering?
19. Interpret the result you obtained