**Fine-tuning Wav2Vec2 for English ASR with 🤗 Transformers**

In our project we follow and implement [this](https://colab.research.google.com/drive/1FjTsqbYKphl9kL-eILgUc-bl4zVThL8F?usp=sharing" \l "scrollTo=LBSYoWbi-45k) notebook to learn and understand how the Wav2Vec2 model developed by facebook for automatic speech recogniton (ASR) works/

the notebook use the CTC algorithem (Connectionist Temporal Classification).

CTC algprithem used to train neural networks for sequence-to-sequence problems.

Wav2Vec2CTCTokenizer

**1. Mapping Speech to Context Representations**

* The pre-trained Wav2Vec2 model takes in raw speech audio (like .wav files) and processes it into a sequence of context representations. These representations are dense embeddings that capture the nuances of the speech signal over time, including information about sounds and patterns in the audio.
* The Wav2Vec2 model creates these embeddings using a transformer-based structure, which allows it to capture contextual information in the speech signal, similar to how BERT captures language context in text.

**2. Adding a Linear Layer for Classification**

* In its pre-trained form, Wav2Vec2 produces context representations but does not directly transcribe them into text. For transcription, a linear layer is added on top of Wav2Vec2’s transformer block to classify each context representation into a specific token in the vocabulary.
* This linear layer maps each context embedding to a vocabulary token (like a letter or word), effectively transforming Wav2Vec2 from a general speech model into a transcription model.
* This is analogous to BERT’s usage: after pretraining on general language, a linear layer is added to BERT’s embeddings to fine-tune it for specific tasks, like sentiment analysis or named entity recognition.

**3. Vocabulary Size and Fine-Tuning**

* The output size of the linear layer corresponds to the number of tokens in the vocabulary, which is determined by the labeled transcription dataset used for fine-tuning.
* For instance, if the transcription dataset (like TIMIT) contains specific phonemes, letters, or words, these elements will define the model's vocabulary. This vocabulary allows Wav2Vec2 to output transcriptions that match the labels in the fine-tuning data, regardless of the specific speech sounds it learned during pretraining.

**4. Defining a Custom Vocabulary Based on the Dataset**

* The first step in fine-tuning is to examine the transcription dataset (in this case, TIMIT) and define a vocabulary that matches the dataset’s transcriptions.
* This step ensures that Wav2Vec2 learns to classify each speech segment into tokens that exist in the dataset, allowing it to accurately transcribe the speech signal based on the labeled examples provided during fine-tuning.

**In summary, by adding a linear layer and defining a vocabulary from the dataset, the pre-trained Wav2Vec2 model is fine-tuned to map its context representations directly to textual transcriptions, effectively making it a speech-to-text model for the target dataset.**

1) Loading the timit data set create Wav2Vec2CTCTokenizer:

* timit is a short data set contains only 5 hours but it gives a lot more than just ‘text’ labels and audio files(‘file’), timit also give us the 'phonetic\_detail' but in this notebook the autors wantet the be general as posibble so they chose to use the text only.
* Since the notebook doesn’t use language model we remove special characters such as (,.?!;:) because they don’t correspond to a characteristic sound unit, we also want to use lower characters only.
* the CTC algorithm classify speech chunks into letters, so the next step is the map all distinct letters in train and test dataset into our vocabulary.
* **We want to also map “ “ and ‘ characters so the model will learn to predict when a word ends and to differentiate between words such as “it’s” and “its”**
* **We also add [PAD] to our vocabulary as CTC’s “blank token”, and [UNK] so the model will deal with characters not encountered in the data set.**
* To create a Wav2Vec2CTCTokenizer we save the vocabulary we created as json file and load it is the tokenizer

(tokenizer = Wav2Vec2CTCTokenizer("./vocab.json", unk\_token="[UNK]", pad\_token="[PAD]", word\_delimiter\_token="|"))

**2 Sampling and Sampling Rate**

* **Continuous to Discrete Conversion**: Speech is a continuous signal, but for computers to process it, it needs to be represented in discrete units. This process is known as *sampling*.
* **Sampling Rate**: The sampling rate determines the number of times per second the audio signal is measured or "sampled." For example, a 16 kHz sampling rate means the audio is sampled 16,000 times per second. Higher sampling rates capture more detail of the original sound but result in more data to process.
* **Importance of Consistent Sampling Rates**: ASR models, like Wav2Vec2, are pre-trained on audio sampled at a specific rate. For optimal performance, it is essential to use the same sampling rate during fine-tuning as was used during pre-training. Different sampling rates create different data distributions, so using mismatched sampling rates (e.g., training with 16 kHz and fine-tuning with 32 kHz) can negatively affect model accuracy.
* **Matching Sampling Rates**: Since Wav2Vec2 was pre-trained on LibriSpeech and LibriVox data, which have a 16 kHz sampling rate, the fine-tuning dataset (TIMIT in this case) should ideally also use a 16 kHz sampling rate. If the dataset used a different sampling rate, the audio would need to be upsampled or downsampled to match 16 kHz.

**2. Configuring the Wav2Vec2 Feature Extractor**

* **Feature Extractor Parameters**: To prepare audio data for Wav2Vec2, a feature extractor is configured with specific parameters that dictate how the data is processed before being fed into the model. Here’s a breakdown of each parameter:
  + **feature\_size**: This defines the fixed size of each audio feature vector in the input sequence. Since Wav2Vec2 operates on the raw waveform (not precomputed features like Mel spectrograms), feature\_size is set to 1, indicating that each sample in the sequence is a single value from the raw audio signal.
  + **sampling\_rate**: Specifies the rate at which the audio data was sampled. This should be set to 16 kHz, as this was the rate used during Wav2Vec2’s pre-training.
  + **padding\_value**: To handle batches of audio with varying lengths, shorter sequences are padded to match the longest sequence in the batch. The padding\_value parameter defines what value to use for padding. This ensures uniform input size for batch processing.
  + **do\_normalize**: This determines whether the audio data should be normalized to have zero mean and unit variance. Normalizing audio data can improve model performance by standardizing the input, reducing variability in volume and amplitude.
  + **return\_attention\_mask**: The attention mask specifies which parts of the input are meaningful and which are padding. Wav2Vec2 generally achieves better performance without an attention mask due to specific design choices in its "base" model, but other ASR models typically use an attention mask. If fine-tuning with the larger "large-lv60" Wav2Vec2 model, however, an attention mask should be enabled.

**Summary**

This section emphasizes the importance of maintaining consistency in sampling rates and configuring the feature extractor to align with Wav2Vec2’s pre-training setup. These steps ensure that audio data is correctly processed and represented, making it compatible with the Wav2Vec2 model for fine-tuning on new transcription tasks.