RowanUniversity

Fine-tuning Wav2Vec2 for English ASR with 🤗 Transformers

Prepared for

Dr. Silvija Kokalj-Filipovic

DS 02695

By

Karan Narad, Hriday Mistry

November 16, 2023

Introduction

Automatic Speech Recognition (ASR) has become a cornerstone of modern technology, enabling machines to understand and process human speech. This capability is pivotal in various applications, from voice-activated assistants to accessibility features for those with disabilities. The advent of deep learning has significantly advanced ASR, leading to more accurate and versatile systems.

In this context, Wav2Vec2, a state-of-the-art model developed by Facebook AI, has emerged as a key player in ASR. Its ability to directly process raw audio data and learn robust representations makes it highly effective for speech recognition tasks. The use of the 🤗 Transformers library further simplifies the implementation and fine-tuning of such models, providing a user-friendly interface and a range of tools that streamline the development process.

This report focuses on the fine-tuning of Wav2Vec2 for English ASR. It aims to explore how a pre-trained Wav2Vec2 model can be adapted to recognize and transcribe English speech with high accuracy. The specific objectives include assessing the model's architecture, understanding the pre-training, and fine-tuning processes, and evaluating the adjustments necessary for optimizing the model for English ASR.

Literature Review

ASR has been a topic of extensive research, especially with the rise of deep learning and neural networks. Recent advancements have shifted the focus to pre-trained models like Wav2Vec2, which leverage vast amounts of data to learn generalized speech representations. These models have set new benchmarks in ASR, demonstrating remarkable performance across various languages and domains.

Fine-tuning pre-trained models has become a pivotal aspect of recent research. This process involves adjusting a model, already trained on a large and diverse dataset, to perform a specific task. The significance of fine-tuning lies in its ability to transfer learned features to new tasks, often with a smaller amount of task-specific data, reducing the need for extensive computational resources.

Several studies have focused on fine-tuning Wav2Vec2 for ASR, showcasing its versatility and efficiency. These works highlight the model's adaptability to different languages and accents, underscoring the potential of pre-trained models in personalized and localized ASR applications.

Methodology

In our approach to fine-tuning the Wav2Vec2 model for English Automatic Speech Recognition (ASR), we meticulously followed a series of steps, each designed to tailor this advanced pre-trained model to the nuances of English speech. Our process was anchored in a deep understanding of the model's architecture, its pre-training regimen, and the subsequent fine-tuning procedures, facilitated by the 🤗 Transformers library.

Wav2Vec2 Architecture and Pre-Training

Architecture Overview: Central to our methodology was an appreciation of Wav2Vec2's unique architecture. This model is specifically engineered for raw audio data processing and includes:

A convolutional neural network-based feature extraction layer that processes raw audio waveforms to yield latent speech representations.

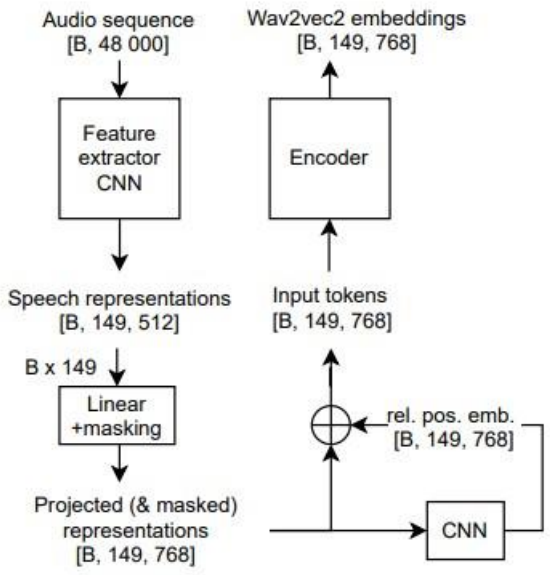
Transformer blocks, equipped with a self-attention mechanism, analyse these representations to grasp the contextual nuances of the audio sequence.

A quantization module discretizes the representations, efficiently capturing various speech elements in a compact vector form.

Pre-Training Insight: The pre-training phase of Wav2Vec2 is instrumental to its capability. During this phase, the model undergoes self-supervised learning, where it learns to predict masked segments of audio. This involves:

Training the model to differentiate between the correct quantized representation of a masked audio segment and various incorrect representations.

Leveraging extensive, diverse, unlabelled audio datasets, which enable the model to learn a broad spectrum of speech patterns and ambient noises.



Fine-Tuning Procedure

Data Preparation:

Dataset Selection: We chose an appropriate English ASR dataset, like TIMIT, known for its balanced representation of English phonetics.

Preprocessing Steps: The audio data was pre-processed to align with Wav2Vec2's input requirements, including adjustments to the sampling rate, segmentation of longer audio files, and normalization of the audio signals.

Model Configuration:

English Language Adaptation: We tailored the model's output layer to suit the phoneme or character set of the English language.

Tokenizer Customization: The tokenizer, essential for converting text transcriptions to a numerical format, was adapted to accommodate English-specific characteristics, ensuring accurate transcription processing.

Hyperparameter Tuning:

Learning Rate Selection: A crucial step was selecting an optimal learning rate, with a preference for a lower rate to enable gradual refinement of the pre-trained weights.

Batch Size and Epochs Determination: The batch size and the number of epochs were set based on the dataset size and the available computational resources.

Optimizer and Scheduler Choice: We opted for an effective optimizer like AdamW and a dynamic learning rate scheduler to manage the learning rate throughout the training process.

Training Strategy:

Training Execution: Employing the Hugging Face Trainer API or a custom training loop, we fine-tuned the model on the prepared English ASR dataset.

Performance Metrics: We implemented evaluation metrics such as the Word Error Rate (WER) to gauge the model's performance on validation and test datasets.

Introduction: Coding Procedure

This report aims to provide a comprehensive explanation of the fine-tuning process for Wav2Vec2's pretrained checkpoints on any English Automatic Speech Recognition (ASR) dataset, all without the need for a language model. The Wav2Vec2 model achieves contextualized speech representations by introducing random masking of feature vectors before feeding them into a transformer network. This approach demonstrates competitive results comparable to state-of-the-art ASR systems, achieved through pretraining and subsequent fine-tuning on a minimal amount of labeled speech data.

To illustrate this process, we conduct fine-tuning on the "base"-sized pretrained checkpoint, leveraging the relatively small Timit dataset comprising just 5 hours of training data. The fine-tuning process employs Connectionist Temporal Classification (CTC), an algorithm specifically designed for training neural networks in sequence-to-sequence problems, particularly in Automatic Speech Recognition and handwriting recognition domains.

To proceed, it is necessary to install both the 'datasets' and 'transformers' packages. Additionally, the 'soundfile' package is required for loading audio files, and 'jiwer' is essential for evaluating the fine-tuned model, specifically using the Word Error Rate (WER) metric.

Tokenizer and Feature Extractor

Automatic Speech Recognition (ASR) models perform the task of transcribing speech into text.

This involves the utilization of a feature extractor to process the speech signal into the model's input format (e.g., a feature vector) and a tokenizer to process the model's output format into text. In 🤗 Transformers, the Wav2Vec2 model is equipped with both a tokenizer, named Wav2Vec2CTCTokenizer, and a feature extractor, known as Wav2Vec2FeatureExtractor.

Creating a Wav2Vec2CTCTokenizer

The pretrained Wav2Vec2 checkpoint translates a speech signal into a series of context representations. When fine-tuning Wav2Vec2, an additional step is needed to link this contextual sequence to its corresponding transcription. To achieve this, we incorporate a linear layer on the top of the transformer block. Similar to how BERT employs a post-pretraining linear layer for classification, this new layer categorizes each context representation into a specific token class. Notably, the size of this layer is determined by the number of tokens in the vocabulary, which is influenced solely by the labeled dataset used for fine-tuning, rather than Wav2Vec2's original pretraining task.

Creating a Wav2Vec2 feature extractor

Wav2Vec2 was pretrained using audio data from LibriSpeech and LibriVox, both sampled at 16kHz. Our fine-tuning dataset, Timit, aligns with this 16kHz sampling rate.
To instantiate a Wav2Vec2 feature extractor object, the following parameters are necessary:
1. feature_size: This represents the dimension of each feature vector in the input sequence. For Wav2Vec2, it's 1 because the model was trained on the raw speech signal.
2. sampling_rate: This denotes the rate at which the model was trained. In this case, it is 16kHz to match the training conditions.
3. padding_value: For batched inference, shorter inputs are padded with a specific value.
4. do_normalize: This parameter indicates whether the input should undergo zero-mean-unit-variance normalization. Typically, speech models perform better with normalized input.
5. return_attention_mask: This parameter determines whether the model should use an attention_mask for batched inference. It's generally advisable for models to utilize attention_mask to mask padded tokens during the inference process.

To keep things simple for users, we bundle the feature extractor and tokenizer of Wav2Vec2 into a single Wav2Vec2Processor class. This way, users only need to create a model and a processor object.

Text Data Preprocessing

Let's begin by loading the dataset. Usually, ASR datasets provide various details for each audio file, but for our fine-tuning on Timit, we're interested in the transcribed text. Timit offers additional information like 'phonetic_detail', 'word_detail', 'dialect_region', 'sentence_type,' but we'll focus solely on the transcriptions.

We've noticed that the transcriptions are quite clean, resembling written text more than spoken dialogue. If there are special characters like ,.?!;: in the transcriptions, we typically exclude them, except for the ' character, crucial for distinguishing words like "it's" and "its." We also ensure all text is in lowercase.

In the CTC approach, where speech chunks are classified into letters, we follow the same principle. We collect all distinct letters from the training and test data to build our vocabulary. A mapping function combines all transcriptions into one long string and then transforms it into a set of characters. It's crucial to use batched=True so that the function has access to all transcriptions simultaneously.

Next, we create a dictionary with enumerated distinct letters from both training and test datasets. We include an "unknown" token for handling characters not encountered in Timit's training set and a padding token corresponding to CTC's "blank token," vital for addressing issues like consecutive character outputs. The "blank token" doesn't signify anything and is simply omitted from the final output.

Audio Data Preprocessing

Wav2Vec2 anticipates the audio file in a 1-dimensional array format. In the initial step, we load all audio files into the dataset object. Various Python libraries, like librosa, soundfile, and audioread, facilitate the reading and processing of audio files in '.wav' format. Typically, an audio file contains both its values and the sampling rate used to digitize the speech signal. We ensure to store

both aspects in the dataset by crafting a map() function accordingly. Now, with the data as a 1-dimensional array, the sampling rate consistently corresponds to 16kHz, and the target text is normalized.

Training

The 🤗 's Trainer class provides an API for feature-complete training in PyTorch for most standard use cases. Utilizing Trainer involves the following key steps:

1. Define a Data Collator: Given Wav2Vec2's substantial input length compared to output length, dynamic padding within training batches is more efficient. This means padding each training sample only up to the length of the longest sample in its batch, not the overall longest sample. Similar to common data collators, padding tokens in labels are set to -100 to exclude them when computing the loss.

2. Evaluation Metric: The model is evaluated on the word error rate (WER) during training. The model outputs a sequence of logit vectors: $y_1,...,y_m$ with $y_1=f_\theta(x_1,\ldots,x_n)[0]$ and $n>>m$, and logit vector y_1 contains the log-odds for each word in the vocabulary we defined. Then we take the argmax() to determine the most likely prediction. Encoded labels are transformed back to the original string, replacing -100 with the pad_token_id, and decoding the ids while ensuring consecutive tokens are not grouped together in CTC style.

3. Load a Pretrained Wav2Vec2 Checkpoint:We need to load a pretrained checkpoint and configure it correctly for training. To conserve GPU memory, we enable PyTorch's gradient checkpointing and set the loss reduction to "mean." Setting requires_grad to False is possible since this part of the model has already undergone sufficient pretraining.

4. Define Training Parameters:
Several parameters related to training are defined:

- 'group_by_length': Enhances efficiency by grouping training samples of similar input length into one batch, reducing the number of unnecessary padding tokens.
- 'learning_rate' and 'weight_decay': Tuned heuristically for stable fine-tuning.

Target final WER should be below 0.3 in our case, considering that state-of-the-art phoneme error rates (PER) are just below 0.1 and WER is typically worse than PER.

Evaluation

We'll utilize the map() function to predict the transcription for each test sample and save the predictions in the dataset. The overall Word Error Rate (WER) is approximately 18.6%, a reasonable result. It's evident that the predicted transcriptions closely resemble the target transcriptions acoustically but often include spelling or grammatical errors. This is expected since we rely solely on Wav2Vec2 without employing a language model. The model demonstrates some invariance to speaking rate, often repeating the same token when the speech chunk corresponds to the same token. This property makes CTC a powerful algorithm for speech recognition, as the transcription is often independent of the speech file's length.

Difficulties

- TIMIT dataset is available at Link for $250, it is not easy to access the data.
- The creators of the 'Fine-Tune Wav2Vec2 for English ASR Notebook' available at Link are utilizing pre-defined functions, namely 'load_dataset' and 'load_metric,' to import audio and text data in the required formats for their project. Unfortunately, the specific details of how these functions achieve this were not evident to us. Our attempts to import audio data using alternative methods resulted in an error message: "file does not start with RIFF id."

References

1. https://www.cs.ru.nl/bachelors-theses/2022/Thomas_Kolb___1027332___Fine-tuning_Wav2vec2.0_on_caption_data.pdf

2. https://arxiv.org/pdf/2109.15053.pdf

3. https://jonathanbgn.com/2021/09/30/illustrated-wav2vec-2.html

4. https://medium.com/@shiryc/from-wav2vec2-to-decoded-sentences-9078e5b56d1f

5. https://aws.amazon.com/blogs/machine-learning/fine-tune-and-deploy-a-wav2vec2-model-for-speech-recognition-with-hugging-face-and-amazon-sagemaker/

6. https://proceedings.neurips.cc/paper_files/paper/2020/file/92d1e1eb1cd6f9fba3227870bb6d7f07-Paper.pdf

7. https://doi.org/10.1016/B978-0-12-802398-3.00001-5.

8. https://colab.research.google.com/drive/1FjTsqbYKphl9kL-eILgUc-bl4zVThL8F

9. https://jalammar.github.io/illustrated-bert/

10. https://distill.pub/2017/ctc/

11. https://huggingface.co/spaces/evaluate-metric/wer

12. https://huggingface.co/docs/transformers/main/model_doc/wav2vec2

13. https://huggingface.co/datasets/timit_asr