NLP Assignment 4: Research

Name: Alireza Dastmalchi Saei

Stu No.: 993613026

Contents

1 Question No. 1

Explain the architecture and pre-training process of wav2vec 2.0.

1.1 Architecture

1.1.1 Feature Encoder

The first stage of Wav2Vec 2.0 is the feature encoder, which transforms raw audio waveforms into latent speech representations. This is achieved using a stack of convolutional neural network (CNN) layers. These layers capture local dependencies within the input waveform and produce a sequence of feature vectors.

1.1.2 Quantization Module

The quantization module discretizes the continuous feature vectors into a finite set of discrete codebook entries. This process creates a set of discrete latent representations, often referred to as quantized representations.

During training, the model learns to predict these discrete latent representations. This step is for reducing the complexity of the data and providing a more manageable form for subsequent processing.

1.1.3 Context Network

This network consists of Transformer layers, which are well-suited for capturing long-range dependencies and contextual information across the sequence of quantized representations.

The context network refines the representations by considering the relationships and dependencies between different parts of the input sequence. This step is for enhancing the model's ability to understand and predict sequences of speech features.

1.1.4 Contrastive Task

The model learns to distinguish between the true quantized representation of a masked part of the input and a set of distractor representations.

By predicting the correct quantized representation despite the masking, the model effectively learns to encode useful information about the audio signal into its latent representations.

1.2 Pre-Training Process

The pre-training of Wav2Vec 2.0 involves learning to predict masked parts of the input audio data based on the surrounding context. The model learns to identify the correct quantized latent audio representation from a set of distractors for each masked time step, and is then fine-tuned on labeled data for specific tasks.

1.2.1 Masking

To pre-train the model, a certain proportion of time steps in the latent feature encoder space are masked. A proportion of the feature encoder outputs, or time steps, are masked before they are fed into the context network. These masked outputs are replaced with a trained feature vector shared across all masked time steps. Masking is done by randomly sampling a certain proportion p of all time steps without replacement to act as starting indices. From each sampled index, M consecutive time steps are masked. The spans of these masked time steps may overlap.

1.2.2 Objective

The training objective during pre-training consists of two parts: a contrastive loss and a codebook diversity loss.

- **Contrastive Loss** (L_m):
  - The model learns speech representations by solving a contrastive task, which requires identifying the true quantized latent speech representation for a masked time step from a set of distractors.
  - Given the context network output c_t centered over a masked time step t, the model must identify the true quantized latent speech representation q_t among $K + 1$ candidates (including K distractors). These distractors are uniformly sampled from other masked time steps within the same utterance.
  - The contrastive loss is defined as:

$$L_m = -\log \frac{\exp(\text{sim}(c_t, q_t))}{\sum_{\tilde{q} \in Q_t} \exp(\text{sim}(c_t, \tilde{q}))}$$

    where $\text{sim}(a, b) = \frac{a^T b}{\|a\| \|b\|}$ is the cosine similarity between context representations and quantized latent speech representations.

- **Diversity Loss** (L_d):
  - The contrastive task relies on the codebook to represent both positive and negative examples. The diversity loss encourages the model to use the codebook entries equally often.
  - This is done by maximizing the entropy of the averaged softmax distribution over the codebook entries for each codebook across a batch of utterances. The entropy H is calculated as:

$$L_d = \frac{1}{GV} \sum_{g=1}^{G} -H(\bar{p}_g) = \frac{1}{GV} \sum_{g=1}^{G} \sum_{v=1}^{V} \bar{p}_{g,v} \log \bar{p}_{g,v}$$

    where G is the number of codebooks and V is the number of entries in each codebook.

The overall pre-training loss is a combination of the contrastive loss and the diversity loss:

$$L = L_m + \alpha L_d$$

where α is a tuned hyperparameter.

1.2.3 Fine-tuning

After pre-training, the model is fine-tuned for specific tasks such as speech recognition:

- A randomly initialized linear projection is added on top of the context network, mapping its outputs to C classes representing the vocabulary of the task.
- The models are optimized by minimizing the Connectionist Temporal Classification (CTC) loss.
- A modified version of SpecAugment is applied during training, which involves masking time-steps and channels to delay overfitting and improve the final error rates, especially on datasets with few labeled examples.

2 Question No. 2

What is the difference between wav2vec 2.0 and wav2vec XLSR-53?

Wav2Vec 2.0 and Wav2Vec XLSR-53 are both speech representation learning models developed by Facebook AI, but the differences are:

2.1 Wav2Vec 2.0

- **Architecture**: Wav2Vec 2.0 consists of three main components: a feature encoder, a quantization module, and a context network.
- **Pre-Training**: The model is pre-trained on large amounts of unlabeled audio data using self-supervised learning. The pre-training involves masking portions of the input audio and learning to predict these masked portions using a contrastive loss.
- **Fine-Tuning**: After pre-training, the model is fine-tuned on labeled data for specific tasks such as automatic speech recognition (ASR).
- **Performance**: Wav2Vec 2.0 has achieved state-of-the-art results on various speech recognition benchmarks, demonstrating its effectiveness in learning high-quality speech representations from raw audio.

2.2 Wav2Vec XLSR-53

- **Architecture**: Wav2Vec XLSR-53 is built on the same architecture as Wav2Vec 2.0 but is designed for cross-lingual speech representation learning.
- **Pre-Training**: XLSR-53 is pre-trained on audio data from 53 different languages, making it capable of learning representations that are useful across multiple languages. This cross-lingual training allows the model to capture universal speech patterns and improve performance on low-resource languages.
- **Fine-Tuning**: Similar to Wav2Vec 2.0, XLSR-53 can be fine-tuned on labeled data for specific ASR tasks, but its pre-training on multiple languages makes it particularly powerful for multilingual applications.
- **Performance**: XLSR-53 has demonstrated strong performance in cross-lingual ASR tasks, particularly in low-resource languages where labeled data is scarce. Its ability to transfer knowledge across languages is a key advantage.

3 Question No. 3

How is decoding performed in the Wav2Vec 2.0 model? Explain the method used.

Decoding in Wav2Vec 2.0 uses a Connectionist Temporal Classification (CTC) decoder. During inference, the model outputs probability distributions over characters for each time step, and the CTC decoder converts these into the most likely sequence of characters, handling the alignment between input speech frames and output text.

- **CTC Loss Function**: During training, the model is fine-tuned using the Connectionist Temporal Classification (CTC) loss function. CTC aligns the input speech frames with the target transcription without requiring explicit frame-level annotations. The CTC loss allows the model to output a probability distribution over possible character sequences for each frame in the input sequence.

  $$\mathcal{L}_{\text{CTC}} = -\log p(y|x)$$

  where y is the target transcription and x is the input speech frames.

- **Beam Search Decoding**: During inference, beam search decoding is used to find the most likely transcription from the CTC output probabilities. Beam search is a heuristic search algorithm that explores multiple possible sequences simultaneously and selects the one with the highest probability.

  - **Beam Width**: The beam width is a parameter that controls the number of sequences explored at each time step. A larger beam width can improve accuracy but increases computational complexity.
  - **Scoring**: The beam search algorithm scores each sequence based on the CTC probabilities and selects the top sequences with the highest scores.

The decoding process in Wav2Vec 2.0, using CTC and beam search decoding (with optional language model integration), enables the model to effectively convert the learned speech representations into accurate text transcriptions. This combination of techniques ensures that the model can handle the variability and complexity of natural speech while producing high-quality transcriptions.

4 Question No. 4

What method or technique do you recommend to get better results?

Alignment Technique: The alignment between input speech frames and output text is handled by the CTC loss function. The CTC loss allows the model to learn the alignment implicitly by considering all possible alignments during training and summing their probabilities, enabling end-to-end training without the need for pre-segmented data.

Language Model Integration: To further improve the accuracy of the transcription, a language model (LM) can be integrated during the decoding process. The LM helps to bias the decoding towards more linguistically probable sequences.

Hyper-parameter Tuning and Scheduling: Adjusting the learning rate, batch size, and other hyper-parameters can significantly impact the performance of the model. Using a learning rate scheduler can also help in finding the optimal learning rate during training.

Data Augmentation: Applying data augmentation techniques such as adding noise, changing pitch, or speed of the audio can make the model more robust to variations in the input data.

More Training Data: Increasing the amount and diversity of the training data can lead to better generalization and improved performance of the model.

Fine-tuning with Domain-specific Data: Fine-tuning the model on domain-specific data can help in adapting the model to the particular characteristics and vocabulary of the target domain, resulting in better performance.

5 Code Report

5.1 Process

First, the dataset is downloaded using the HuggingFace API Token to access the Mozilla Common Voice Dataset. The dataset is then cleaned by removing unnecessary columns and filtering audio durations (**Train**: 4-6 seconds, **Test**: under 15 seconds). Next, text preprocessing involves ignoring specified characters and mapping all characters to a standard set (length: 35). Text normalization follows, and a vocabulary dictionary is created for use in the `Wav2Vec2CTCTokenizer`. Additionally, 5 special characters (<s>, </s>, <pad>, <unk>, and — as delimiter) are added to the dictionary. Audio is resampled from 48000 kHz to 16000 kHz to match the Wav2vec2 model requirements.

The `Wav2Vec2CTCTokenizer`, `Wav2Vec2FeatureExtractor`, and combined processor are defined. The `prepare_dataset` function extracts input features (audio signals) using the processor, and generates corresponding text labels for the model.

A custom `DataCollator` is defined to handle batching and padding of input and label sequences, ensuring uniform sequence lengths for efficient computation. WER (Word Error Rate) is used as an evaluation metric, measuring the percentage of incorrectly predicted words through substitutions, deletions, and insertions relative to the total number of words in the reference transcription.

The `facebook/wav2vec2-large-xlsr-53` model is loaded with specified parameters, and the feature extractor weights are frozen. A `Trainer` is instantiated with elevated evaluation steps to reduce training time. Training commences, followed by evaluation. Key parameters include:

Parameter	**Value**
Batch Size	12
Num Epochs	5
Training Files	2217
Testing Files	5212
Training Time (minutes)	19.67
Total Parameters	315,479,720
Trainable Parameters	311,269,544
WER (Word Error Rate)	1.0

Table 1: Summary of Training Parameters

Finally, the trained model, tokenizer, and feature extractor are saved on HuggingFace at AlirezaSaei/wav2vec2-large-xlsr-persian-fine-tuned.

5.2 Results

During the training process, the predicted sentence lengths progressively decrease until the model predicts empty strings. However, the loss continues to decrease with each training step. This phenomenon is a known behavior in fine-tuning tasks.The typical stages of training can be summarized as follows (under the assumption that the ratio of trainable parameters to training data is appropriate):

- **Beginning**: The model outputs random characters.
- **Early Stages**: The model often outputs nothing, resulting in empty strings.
- **After a While**: The model starts to produce more relevant characters and phrases.

In this particular task, the number of epochs (5) is insufficient to achieve optimal results. Moreover, there is an issue with the speech data filtering by length. The training data consists of samples between 4 and 6 seconds, while the test data includes samples up to 15 seconds. Consequently, the model is likely to perform better on test data that falls within the 4 to 6-second range. This discrepancy in data length filtering could lead to suboptimal model performance on longer test samples.

To address these issues, it may be beneficial to:

1. Increase the number of training epochs to allow the model more time to learn and generalize from the training data.
2. Ensure consistency in the length of training and test data to avoid performance discrepancies.
3. Consider augmenting the dataset with a more diverse range of audio lengths to improve the model's robustness and performance on varying test samples.