Deep Learning for Speech Signals – H.W 5

Submitters:

Lidor Zino I.D: 313551186

Suf Rubinstein I.D: 205853633

Date: 12/02/2025

1. WER and CER

Below is an example of how you might present the Word Error Rate (WER) and Character Error Rate (CER) results after running the code the given dataset. These numbers are also shown in the notebook. The actual numbers will depend on the specific data and training procedure.

Decoder Configuration	Average WER	Average CER
Greedy Decoder	0.459	0.124
CTC Decoder with LM (beam size: 1)	0.911	0.895
CTC Decoder with LM (beam size: 50)	0.064	0.055
CTC Decoder with LM (beam size: 500)	0.064	0.055
CTC Decoder without LM (beam size: 1)	0.787	0.757
CTC Decoder without LM (beam size: 50)	0.032	0.029

Decoder Configuration	Average WER	Average CER
CTC Decoder without LM (beam size: 500)	0. 032	0. 029

These numbers are for the current database. After you run a different training/evaluation loop, the script will print the average WER and CER for each decoder in place of these placeholder values.

2. Description of the Acoustic Model

The acoustic model in this code is built around a pretrained wav2vec2 backbone from the torchaudio.pipelines.WAV2VEC2_ASR_BASE_960H bundle. The steps are:

1. **Pretrained Wav2Vec2 Feature Extractor**
    - The wav2vec2 model is responsible for processing raw audio waveforms (sampled at 16 kHz) and extracting high-level speech representations. It has been pretrained on large-scale audio datasets, which gives it robust general speech features.
2. **Freezing Parameters**
    - All parameters of the pretrained model are frozen, meaning they do not update during backpropagation. This prevents overfitting on small datasets and speeds up training (fewer parameters to optimize).
3. **Adapter (Linear Layer)**
    - A small, trainable **linear adapter** is added on top of the pretrained wav2vec2 model's output. This adapter layer has num_tokens output units, corresponding to the number of symbols (including the blank symbol) in your character set.
    - It maps the 29-dimensional output of the wav2vec2 model to your custom token vocabulary, effectively acting as a classifier head for the CTC objective.

Hence, the entire pipeline is:

Waveform (16 kHz) --> Frozen wav2vec2 feature extractor --> Trainable linear adapter --> CTC loss/decoding

3. Instructions for Running the Code

First, the best way is to get here: *https://colab.research.google.com/drive/1sBTp9FylFwxoBsr5KespYiD3J9TpHbaa*

which is the original code. You will need to upload the data to the colab if you're using that link.

1. **Install Required Libraries**
   Make sure you have Python 3.7+ and the libraries (pip installs) that are in a comment at the beginning of the code in the python file. Also run the commands under #kenlm. Note that the code creates lm.arpa by itself, so you don't have to use the attached one

2. **Prepare Your Data**

   - Organize your training audio files under a folder named train and testing audio files under test (or adjust paths in the code as needed).

   - The script assumes a structure like:
     /content

```
├── train
│   ├── audio_file1.wav
│   ├── audio_file2.wav
│   └── ...
├── test
│   ├── test_file1.wav
│   ├── test_file2.wav
│   └── ...
├── lm.arpa
└── lexicon.txt
```

   - Make sure lm.arpa (the ARPA language model file) and lexicon.txt (mapping of words to tokens) are placed in the same root directory (/content in the code).

3. **Adjust Hyperparameters & Paths**

   - Inside the code, verify that DATA_ROOT points to the correct directory (e.g., '/content').
   - Adjust LEARNING_RATE, split_ratio, or other hyperparameters as desired.