

11-751/18-781 Speech Recognition and Understanding (Fall 2025)

Coding Assignment 4

OUT: November 5th, 3:30 PM ET

DUE: November 21th, 3:30 PM ET

Instructor: Prof. Shinji Watanabe

TAs: Jinchuan Tian, Shikhar Bharadwaj, Masao Someki, Chin-Jou Li

Collaboration Policy

Assignments must be completed individually. You are allowed to discuss the homework assignment with other students and collaborate by discussing the problems at a conceptual level. However, your answers to the questions and any code you submit must be entirely your own. If you do collaborate with other students (i.e. discussing how to attack one of the programming problems at a conceptual level), you must report these collaborations. Your grade for Coding Assignment 4 will be reduced if it is determined that any part of your submission is not your individual work.

Collaboration without full disclosure will be handled in compliance with CMU Policy on Cheating and Plagiarism: <https://www.cmu.edu/policies/student-and-student-life/academic-integrity.html>

Late Day Policy

You have a total of **5 late days (i.e., 120 hours)** that you can use over the semester for the assignments. If you do need a one-time extension due to special circumstances, please contact the instructor (Shinji Watanabe) via Piazza.

Attention-based Encoder-Decoder for Speech Recognition (5 pts)

Problem Overview

In this assignment, you will implement an Attention-based Encoder-Decoder (AED) with Conformers for end-to-end automatic speech recognition (ASR). Specifically, you will train a model that can identify and recognize speech in two languages: English and Italian. After completing the code, you will train and decode models using the provided train/dev/test data. Finally, you need to submit the generated hypotheses of the blind test set to GradeScope. We will enable the leaderboard, which is based on the word error rate (WER) and language prediction accuracy (ACC) of your models.

Gradescope assignment: <https://www.gradescope.com/courses/1100161/assignments/6586019>

Task: Implement a Conformer-based AED model, train it on the provided data, and decode the trained model on the blind test data. Use the provided template in the handout, and add your code to the template as directed. You can also implement additional features by yourself (e.g., new data augmentation, other ASR architectures, language model decoding, etc), but you need to submit all the code.

Your GradeScope submission: Submit all the code and the generated hypotheses for the blind test set: `decoded_hyp_monolingual.txt` for Checkpoint 1 and `decoded_hyp_bilingual.txt` for Checkpoints 3/4. Note that with the exception of Checkpoint 2, your implementation is not autograded. We have prepared a `prepare_submission.sh` to generate the submission ZIP file.

Grading scheme: Your grade is dependent on 4 checkpoints:

1. 1 point: English-only WER on the blind test set. Your WER must be below 60%.
2. 1 point: Correctly pre-pending the language identification tag in the data loader.
3. 2 points: WER on the bilingual test set. The target threshold will be released later.
4. 1 point: Language identification ACC on the bilingual test set. The target threshold will be released later.

Checkpoints 1 and 2 are designed to check your implementation. You should be able to achieve the 60% WER target for Checkpoint 1 with little-to-no hyper-parameter tuning. No partial credit is available for these two checkpoints.

You can receive partial credit for Checkpoints 3 and 4.

If the bilingual WER is greater than 42%, you will earn no points for Checkpoint 3. If the WER is less than 30%, you will earn 2 points. If the WER is within the interval $[30, 42]$, the grade will be linearly interpolated between 2 and 0, and it will be rounded to have a single digit.

If ACC is less than 85% you will earn no points for this metric. If ACC is greater than 90%, you will earn 1 point. If the ACC is within the interval $[85, 90]$, the grade will be linearly interpolated between 1 and 0, and it will be rounded to have a single digit.

We also enable the leaderboard for you to compete with each other. If your bilingual WER is less than 30% *and* ranks top 10, you can receive 1 bonus point.

Coding Instructions

Set up python environment

Suppose you use anaconda to manage python environments. You can first create a new environment and install the required packages.

```
# create a new environment
conda create --name 11751-fall25-coding4 python=3.11
conda activate 11751-fall25-coding4
```

```
# install pytorch
pip install torch==2.6.0+cu126 torchaudio --index-url https://download.pytorch.org/whl/cu126
```

```
# install other packages
pip install -r requirements.txt
```

You can also think of using `pixi` and `uv` if you have already installed them:

```
# create a new environment
pixi global install ffmpeg
uv venv -p 3.11
source .venv/bin/activate
```

```
# install pytorch
uv pip install torch==2.6.0 torchaudio==2.6.0
```

```
# install other packages
uv pip install -r requirements.txt
```

We specifically prepared a setup script for the `pixi` and `uv` settings. You can just run the setup function and start training your models.

```
. setup_uv.sh
```

If you encounter `libsndfile` error, install it via conda-forge or `pixi`.

Data

The dataset is stored in a shared directory on PSC Bridges-2 and has a size of 20GB. To save space, please create a symbolic link instead of copying it to your own directory, if you would like to use PSC for this assignment. If you do not have access to PSC, make sure you follow the instructions in <https://piazza.com/class/mer2z78ce504p4/post/119> as soon as possible.

Note that you can use any computing resource that you have access to. You can download the data to your own machine using `scp`, `rsync` or any tool you prefer.

```
# our data is here: /ocean/projects/cis250187p/shared/coding4
```

```
# put your code somewhere and go to the code directory
# then create a symbolic link of the data to the current directory
ln -s /ocean/projects/cis250187p/shared/coding4/dump .
```

This directory contains the sub-directory `dump`, which contains six dataset sub-directories: `train_bilingual`,

`dev_bilingual`, `test_bilingual`, `train_monolingual`, `dev_monolingual` and `test_monolingual`. The json files in each directory (e.g., `dev_bilingual/data.json`) contain the information about the input audio and transcripts. More specifically, each utterance has `input` and `output` keys, which contain the path to the audio files and raw transcripts, respectively. The json file for the blind test set only contains the input speech features but has no transcript. Please check these json files to better understand the data. For the tokenizations, a text file which is the list of BPE tokens (`dict.txt`) and the BPE model used to obtain those tokens from raw text (`bpe.model`) is provided in the zip file.

Code template

The code template has the following structure. **You need to complete the “TODO” items.**

1. `conf`: The training and decoding configs are saved here. We use the `yaml` format for configs, but you can also pass the arguments in command line. A sample training config is provided as `base.yaml`. The training and decoding arguments are defined in `train.py` and `decode.py`, respectively.
2. `data`: This is the data directory which is already prepared. Note that this is NOT distributed in the code template. Please read the previous section for more information.
3. `models`: This directory contains the definitions of various modules.
 - (a) `asr_model.py`: Defines the overall ASR model. You need to finish the missing parts.
 - (b) `frontend.py`: Extract log-Mel filterbank speech features from raw audio.
 - (c) `encoder.py`: Defines a single encoder layer and the entire Transformer encoder. You need to finish the missing parts.
 - (d) `decoder.py`: Defines a single decoder layer and the entire Transformer decoder.
 - (e) `layers.py`: Has various types of basic building blocks used in other scripts. You need to finish the `forward` functions of `PositionwiseFeedForward` and `MultiHeadedAttention`.
 - (f) `scheduler.py`: Defines the warmup scheduler which is typically used to train Transformers.
4. `decode.py`: Interface for decoding a trained model. You can use it to decode any set which computes the WER. You need to run it for the blind `test` set to generate the hypotheses `decoded_hyp.txt`, which needs to be submitted to Gradescope.
5. `loader.py`: Has the utility functions to generate mini-batches and data loaders. You need to finish the `tokenize_text` function to support language identification tags. This corresponds to Checkpoint 2.
6. `specaug.py`: Provides the functions for Spec Augment. Using this is optional, but you need to apply it yourself.
7. `train.py`: Interface for training a model.
8. `trainer.py`: Defines the `Trainer` class which performs training, validation etc.
9. `utils.py`: Has utility functions that are used by other scripts.

After implementing all missing functions, you can modify the training config as you see fit and start training by executing the following:

```
python train.py --tag train_tag
```

where `--tag train_tag` is a suffix of your experiment directory. You can specify it to distinguish between different trials/runs.

On the command line, you may see logs like the following, and it might appear as though the process is stuck:

```
[info] #Utts: 37033 | Created 1727 minibatches containing 14 to 124 samples, and on average 21 samples
[info] #Utts: 3677 | Created 101 minibatches containing 13 to 119 samples, and on average 36 samples
[info] Minibatches have been randomly shuffled.
[info] Minibatches have been randomly shuffled.
[info] Minibatches have been randomly shuffled.
[info] Minibatches have been randomly shuffled.
```

However, this is expected behavior. The actual training progress is logged in the file: `exp/train_tag/logs/train.log`. These messages are printed because the minibatches are reshuffled at the beginning of each epoch.

To switch between monolingual and multilingual training, you can use the `--mode` flag.

```
python train.py --tag train_mono --mode monolingual
python train.py --tag train_multi --mode multilingual
```

This will determine if the language identification tags should be pre-pended or not. You can also use the `--train_json` and `--valid_json` flags to change the datasets used.

```
python train.py --tag train_mono --mode monolingual \
  --train_json dump/raw/train_monolingual/data.json \
  --valid_json dump/raw/dev_monolingual/data.json
```

With a single GPU, the training usually takes (less than) one day, depending on your model config and GPU capacity. Based on our experience, a single GPU should be sufficient for this dataset. Please do not use multiple GPUs, as the code is not optimized nor tested for distributed training. It will likely be less efficient than training on a single GPU.

To decode a trained model on the monolingual/bilingual datasets, execute the command:

```
python decode.py --exp_dir your_exp_dir \  
  --ckpt_name your_ckpt --decode_tag your_custom_tag \  
  --recog_json dump/raw/test_monolingual/data.json --mode monolingual  
  
python decode.py --exp_dir your_exp_dir \  
  --ckpt_name your_ckpt --decode_tag your_custom_tag \  
  --recog_json dump/raw/test_bilingual/data.json --mode multilingual
```

You can find all supported arguments in these python scripts. Please try to understand them before running the job.

NOTE: If you use PSC, you will need `sbatch` to submit your actual jobs to GPU or CPU partitions. It is NOT allowed to run these jobs on a login node. Please refer to the weekly assignment material and the official documentation of PSC for example usages.

Conformer

Please refer to the original paper for more details about Conformer with the self-attention mechanism: <https://arxiv.org/abs/2005.08100>

Here, we briefly review the concepts and connect them with the ASR task. A speech feature sequence (e.g., logMel) is first processed by a convolutional module which extracts low-level features and also downsamples the input along the time dimension. Then, positional embeddings are added to the sequence, which is important for Conformer layers. The subsequent Conformer encoder is a stack of multiple blocks. Each block consists of a multi-head self-attention module, a convolution module, and a position-wise feed-forward network. The residual connection, layer norm and dropout are applied to all of them. The output of the Conformer encoder is a high-level feature sequence, which is used to calculate the attention loss. The Transformer decoder works in a similar manner, except that it takes discrete token IDs as input and predicts the conditional probability distribution over next tokens.

As noted above, you need to finish the implementation of `MultiHeadedAttention`, which is used in both the encoder and decoder. Equation (1) in the paper linked above may be helpful.

Multilingual Speech Recognition

Multilingual ASR models are those that function on multiple languages¹. There are different advantages to training models multilingually, but two common reasons are:

1. Efficiency - You only need to train and tune hyper-parameters for a single model. You also only need to host and deploy a single model to production.
2. Transfer Learning - Multilingual training can help improve performance on low-resource languages.

A common realization of multilingual ASR is through prompt-based joint language identification (LID) and ASR. This effectively trains the model to perform both LID and ASR in a hierarchical manner. The model first predicts an LID token, which is then used as a condition to predict the ASR transcript. A convenient part of this technique is that it only requires modifying the text data used to train the model, otherwise it is completely identical to the AED formulation. An example is shown below:

```
a_typical_asr_utterance I WANT TO GO TO THE CMU CAMPUS  
asr_utterance_with_lid_ [ENG] I WANT TO GO TO THE CMU CAMPUS
```

¹Typically this means only one language per utterance. More than one language per utterance implies code-switching, which is related but ultimately different

different_lid_utterance [ITA] VOGLIO ANDARE AL CAMPUS DELLA CMU

Many large-scale multilingual ASR models, such as Whisper or OWSM, use this style of training. For this assignment, you will need to prepend [ENG] and [ITA] language tags to your model's target labels, in order to distinguish between English and Italian.

General steps to build a neural network in PyTorch

To build a neural network model in Pytorch, we have the following important steps:

1. **Data Preparation:** Prepare the data by extracting speech features and tokenizing the text transcript based on the unit you use to model speech. Then compile this data into easy readable format like json files. Partition the data for training, development and evaluation. All the above steps have been done for you. For training, and development, you will have access to the groundtruth text transcript labels and speech features. For test, you will use speech features to obtain predictions of the text transcript from your model.
2. **Data Loading and Batching:** Build a PyTorch Dataset object that has the `__getitem__` and `__len__` methods, which return a data sample given an element key, and the total number of data samples in a partition, respectively. Then we create a DataLoader with a custom batching mechanism. We create adaptive batches of examples based on the input size. This means that we can have batches of different sizes such that the total number of input feature floats (`batch_bins`) has a maximum for each batch. If you encounter GPU out-of-memory errors, please reduce the value of `batch_bins`.
3. **Trainer:** We need a trainer that performs the training steps. Within each epoch of training, we have training and validation. In the train step, we load data from the data loader, run it through our model, compute the loss, and then perform back propagation. Based on the computed gradients, we update the model parameters by calling the `step` method of the optimizer. In the validation step, we load the validation data, do forward propagation through the model, and compute statistics including loss. After the training and validation steps, we log statistics for the epoch, and save models.
4. **Neural Network Definition:** This is the focus of the current assignment. Neural networks are written in PyTorch using `torch.nn`, which contains many standard neural network layers including Linear, Convolutions, RNN, LSTM etc. Each submodule is written as a `torch.nn.Module` which has an `__init__` and `forward` method. The initialization method defines the neural network layers within the module, and the forward method describes how forward propagation will occur given the input using the neural network layers defined in the `__init__` function.
5. **Decoding and Search:** After training the sequence model, you will use the trained model to produce text transcriptions for an unknown test set. Hence, this would necessitate writing a decode function. Typically, beam search is used, but for this assignment, you will implement greedy search, which is easier. Please feel free to try beam search if you are interested.

How to improve performance

We share several tips to improve the performance (i.e., reduce the WER) of this Transformer ASR model.

1. Tune the hyperparameters, especially the learning rate, warmup steps, epochs.
2. Use wider and/or deeper models. You can increase the hidden dimension and/or the number of encoder blocks.
3. Apply a data augmentation, such as SpecAugment. We have provided an implementation in `specaug.py`.

4. Implement more advanced architectural changes, such as Branchformer. We provide some basic building blocks, such as `RelPositionMultiHeadedAttention`, `RelPositionalEncoding`, and `ConvolutionModule`. You can also directly use these parts by modifying your configuration file.
5. Train a separate language model and perform beam search.
6. Add CTC Loss to your training objective.

Example hypothesis file

Your generated `decoded_hyp_bilingual.txt` should have the following format (the utterance ids and the text will be different since this example is from the dev set):

```
1272-128104-0000 [ENG] MISTER CQLOOR AS THE AP IMPOUSSIL OF THE MIDDLE CLASSES AND WE
1272-128104-0001 [ENG] NOR IS MISTER CULTS MANNERLESS INTERESTING THANN HIS METTER
1272-128104-0002 [ENG] HE TELLS US THAT AT THIS FESTIVEB SEASON OF THE YEAR WITH
1272-128104-0003 [ITA] HA G NON FARE QUANDO SIRFREDERIQ LATENS IL LAVORO È PRONTO
```

Similarly, your generated `decoded_hyp_monolingual.txt` should have the following format.

```
1272-128104-0000 MISTER CQLOOR AS THE AP IMPOUSSIL OF THE MIDDLE CLASSES AND WE
1272-128104-0001 NOR IS MISTER CULTS MANNERLESS INTERESTING THANN HIS METTER
1272-128104-0002 HE TELLS US THAT AT THIS FESTIVEB SEASON OF THE YEAR WITH
1272-128104-0004 HE HAS G DOBTS WH THEIR SIRFREDERIC LATENS WORK IS READY
```