11-751/18-781 Speech Recognition and Understanding (Fall 2022)
Coding Assignment 4

OUT: November 2nd, 4:40 PM ET
DUE: November 18th, 11:59 PM ET

Instructor: Prof. Shinji Watanabe
TAs: Xuankai Chang, Yifan Peng, Jiatong Shi

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 1 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.

End-to-End Automatic Speech Recognition (5 pts)

Problem Overview

In this assignment, you will implement a Transformer-CTC model for end-to-end automatic speech recognition (ASR). 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. Similar to some previous assignments, we will enable the leaderboard which is based on the word error rate (WER).

Gradescope assignment: https://www.gradescope.com/courses/412024/assignments/2403517

Task: Implement a Transformer-CTC 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 `decoded_hyp.txt` of the blind test set. Note that your implementation is not autograded. Your final grade depends on the test WER.

Grading scheme: Your grade depends on the WER of the blind test set. The full grade is 5 points. If the WER is greater than 30%, the grade will be 0. If the WER is less than 15%, the grade will be 5. If the WER is within the interval $[15, 30]$, the grade will be linearly interpolated between 5 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 WER is less than 15% and ranks top 10, you can receive 1 bonus point. If you achieve the lowest WER (rank top one), you can get another 1 bonus point (i.e., totally 2 bonus points).

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 e2e-asr python=3.9
conda activate e2e-asr

# install pytorch
conda install pytorch==1.12.1 torchaudio==0.12.1 cudatoolkit=11.6 -c pytorch -c conda-forge

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

Data

The dataset is stored in a shared directory on PSC Bridges-2, which has a size of 9GB. 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.

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/tra220029p/pyf98/coding4/data

# 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/tra220029p/pyf98/coding4/data ./
```

This data directory contains three sub-directories: `train_sp`, `dev` and `test`. It also contains a text file which is the list of BPE tokens: `unigram300_units.txt`. The logMel features are already extracted and stored in the Kaldi format. The json files in each directory (e.g., `data/train_sp/data_unigram300.json`) contain the information about features and transcripts. More specifically, each utterance has `input` and `output` keys, which contain the path to Kaldi ark files, and BPE-level tokenized 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.

Code template

The code template has the following structure:

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 by us. 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. **You need to complete the "TODO" items**.
   (a) `asr_model.py`: Defines the overall ASR model. You need to finish the missing parts.

(b) `ctc.py`: Defines the CTC module which has a linear layer and can compute CTC loss. You need to finish the missing parts.

(c) `encoder.py`: Defines a single encoder layer and the entire Transformer encoder. You need to finish the missing parts.

(d) `layers.py`: Has various types of basic building blocks used in other scripts. You need to finish the `forward` function of `PositionwiseFeedForward`.

(e) `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.
6. `spec_augment.py`: Has the functions to perform spec augmentation. However, it is not used by default. If you want to try it as a data augmentation technique, please modify the corresponding part based on your understanding.
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
```

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. If you really want to use multiple GPUs, please note that the current code template only supports (actually not tested) DataParallel but not DistributedDataParallel. The former is known to be much less efficient than the latter.

To decode a trained model, execute the command:

```
python decode.py --exp_dir your_exp_dir --ckpt_name your_ckpt --decode_tag your_custom_tag
```

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.

Transformer-CTC

Please refer to the original paper for more details about Transformers with the self-attention mechanism: https://arxiv.org/abs/1706.03762

Here, we briefly review the concepts and connect them with the ASR task. Figure 1 shows the typical architecture of Transformer with a CTC loss for ASR. 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 Transformer layers. The subsequent Transformer encoder is a stack of multiple blocks. Each block consists

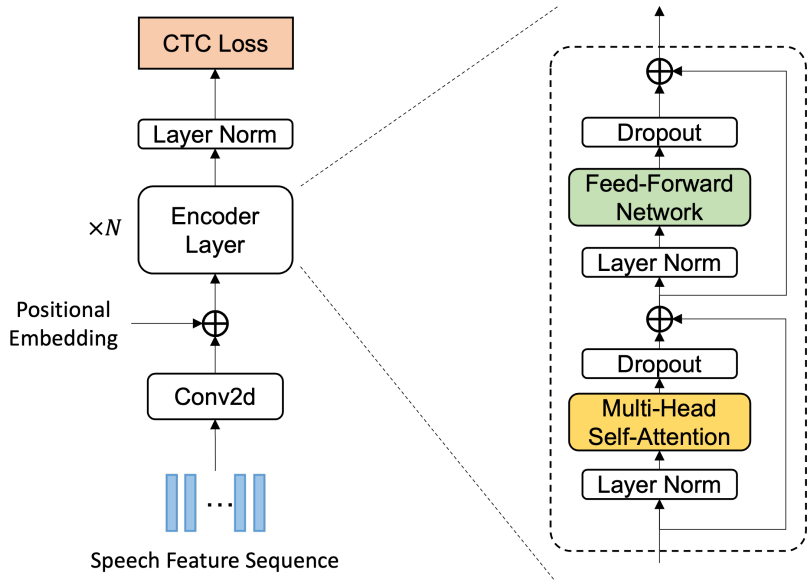

Figure 1: Transformer architecture.

of a multi-head self-attention module and a position-wise feed-forward network. The residual connection, layer norm and dropout are applied to both of them. The output of the Transformer encoder is a high-level feature sequence, which is used to make predictions and calculate the CTC loss.

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.

3. **Trainer**: We need to implement 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 and Word Error Rate (WER). 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. As we are building a CTC-based model, we have a major component, i.e., the encoder. Each submodule is written as a `torch.nn.Module` which has an `__init__` and `forward` method. The former defines the important neural network layers within the module, and the forward method describes how forward propagation will occur through the neural network given the input to find the outputs 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 has similar performance if you do not have a separate language model (LM). For CTC greedy search, you just take the most likely token at each time step, merge repeated tokens, and remove blank.

How to improve performance

We share several tips to improve the performance (i.e., reduce the WER) of this Transformer-CTC 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 data augmentation. You can try Spec Augment first.
4. Implement more advanced encoder architectures. We recommend the Conformer [1] architecture, which has been widely used in the speech field.
5. Train a separate language model and perform beam search.

Example hypothesis file

Your generated `decoded_hyp.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 MISTER CQLOOR AS THE AP IMPOUSSIL OF THE MIDDLE CLASSES AND WE ARE GLAD
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 CHRISTMS
1272-128104-0003 HE HAS G DOBTS WH THEIR SIRFREDERIC LATENS WORK IS READY
```

[1] https://arxiv.org/abs/2005.08100