icefall

Release 0.1

icefall development team

Sep 25, 2023

CONTENTS:

Documentation for icefall, containing speech recognition recipes using k2.

CHAPTER

ONE

ICEFALL FOR DUMMIES TUTORIAL

This tutorial walks you step by step about how to create a simple ASR (Automatic Speech Recognition) system with Next-gen Kaldi.

We use the yesno dataset for demonstration. We select it out of two reasons:

- It is quite tiny, containing only about 12 minutes of data
- The training can be finished within 20 seconds on `CPU`.

That also means you don't need a `GPU` to run this tutorial.

Let's get started!

Please follow items below **sequentially**.

Note: The *Data Preparation* runs only on Linux and on macOS. All other parts run on Linux, macOS, and Windows.

Help from the community is appreciated to port the *Data Preparation* to Windows.

1.1 Environment setup

We will create an environment for Next-gen Kaldi that runs on `CPU` in this tutorial.

Note: Since the yesno dataset used in this tutorial is very tiny, training on `CPU` works very well for it.

If your dataset is very large, e.g., hundreds or thousands of hours of training data, please follow *Installation* to install icefall that works with `GPU`.

1.1.1 Create a virtual environment

```
virtualenv -p python3 /tmp/icefall_env
```

The above command creates a virtual environment in the directory `/tmp/icefall_env`. You can select any directory you want.

The output of the above command is given below:

```
Already using interpreter /usr/bin/python3
Using base prefix '/usr'
New python executable in /tmp/icefall_env/bin/python3
Also creating executable in /tmp/icefall_env/bin/python
Installing setuptools, pkg_resources, pip, wheel...done.
```

Now we can activate the environment using:

```
source /tmp/icefall_env/bin/activate
```

1.1.2 Install dependencies

Warning: Remeber to activate your virtual environment before you continue!

After activating the virtual environment, we can use the following command to install dependencies of icefall:

Hint: Remeber that we will run this tutorial on CPU, so we install dependencies required only by running on CPU.

```
# Caution: Installation order matters!

# We use torch 2.0.0 and torchaduio 2.0.0 in this tutorial.
# Other versions should also work.

pip install torch==2.0.0+cpu torchaudio==2.0.0+cpu -f https://download.pytorch.org/whl/
↪torch_stable.html

# If you are using macOS or Windows, please use the following command to install torch␣
↪and torchaudio
# pip install torch==2.0.0 torchaudio==2.0.0 -f https://download.pytorch.org/whl/torch_
↪stable.html

# Now install k2
# Please refer to https://k2-fsa.github.io/k2/installation/from_wheels.html#linux-cpu-
↪example

pip install k2==1.24.3.dev20230726+cpu.torch2.0.0 -f https://k2-fsa.github.io/k2/cpu.html

# Install the latest version of lhotse

pip install git+https://github.com/lhotse-speech/lhotse
```

1.1.3 Install icefall

We will put the source code of icefall into the directory /tmp You can select any directory you want.

```
cd /tmp
git clone https://github.com/k2-fsa/icefall
cd icefall
pip install -r ./requirements.txt
```

```
# Anytime we want to use icefall, we have to set the following
# environment variable

export PYTHONPATH=/tmp/icefall:$PYTHONPATH
```

Hint:

If you get the following error during this tutorial:

```
ModuleNotFoundError: No module named 'icefall'
```

please set the above environment variable to fix it.

Congratulations! You have installed icefall successfully.

1.1.4 For the more curious

icefall contains a collection of Python scripts and you don't need to use `python3 setup.py install` or `pip install icefall` to install it. All you need to do is to download the code and set the environment variable `PYTHONPATH`.

1.2 Data Preparation

After *Environment setup*, we can start preparing the data for training and decoding.

The first step is to prepare the data for training. We have already provided prepare.sh that would prepare everything required for training.

```
cd /tmp/icefall
export PYTHONPATH=/tmp/icefall:$PYTHONPATH
cd egs/yesno/ASR

./prepare.sh
```

Note that in each recipe from icefall, there exists a file `prepare.sh`, which you should run before you run anything else.

That is all you need for data preparation.

1.2.1 For the more curious

If you are wondering how to prepare your own dataset, please refer to the following URLs for more details:

- https://github.com/lhotse-speech/lhotse/tree/master/lhotse/recipes

  It contains recipes for a variety of dataset. If you want to add your own dataset, please read recipes in this folder first.

- https://github.com/lhotse-speech/lhotse/blob/master/lhotse/recipes/yesno.py

  The yesno recipe in lhotse.

If you already have a Kaldi dataset directory, which contains files like `wav.scp`, `feats.scp`, then you can refer to https://lhotse.readthedocs.io/en/latest/kaldi.html#example.

1.2.2 A quick look to the generated files

`./prepare.sh` puts generated files into two directories:

- `download`
- `data`

download

The `download` directory contains downloaded dataset files:

```
tree -L 1 ./download/

./download/
|-- waves_yesno
`-- waves_yesno.tar.gz
```

Hint: Please refer to https://github.com/lhotse-speech/lhotse/blob/master/lhotse/recipes/yesno.py#L41 for how the data is downloaded and extracted.

data

```
tree ./data/

./data/
|-- fbank
|   |-- yesno_cuts_test.jsonl.gz
|   |-- yesno_cuts_train.jsonl.gz
|   |-- yesno_feats_test.lca
|   `-- yesno_feats_train.lca
|-- lang_phone
|   |-- HLG.pt
|   |-- L.pt
|   |-- L_disambig.pt
|   |-- Linv.pt
```

(continues on next page)

(continued from previous page)

```
|   |-- lexicon.txt
|   |-- lexicon_disambig.txt
|   |-- tokens.txt
|   `-- words.txt
|-- lm
|   |-- G.arpa
|   `-- G.fst.txt
`-- manifests
    |-- yesno_recordings_test.jsonl.gz
    |-- yesno_recordings_train.jsonl.gz
    |-- yesno_supervisions_test.jsonl.gz
    `-- yesno_supervisions_train.jsonl.gz

4 directories, 18 files
```

data/manifests:

This directory contains manifests. They are used to generate files in `data/fbank`.

To give you an idea of what it contains, we examine the first few lines of the manifests related to the `train` dataset.

```
cd data/manifests
gunzip -c  yesno_recordings_train.jsonl.gz  | head -n 3
```

The output is given below:

```
{"id": "0_0_0_0_1_1_1_1", "sources": [{"type": "file", "channels": [0],
↪ "source": "/tmp/icefall/egs/yesno/ASR/download/waves_yesno/0_0_0_0_
↪1_1_1_1.wav"}], "sampling_rate": 8000, "num_samples": 50800,
↪"duration": 6.35, "channel_ids": [0]}
{"id": "0_0_0_1_0_1_1_0", "sources": [{"type": "file", "channels": [0],
↪ "source": "/tmp/icefall/egs/yesno/ASR/download/waves_yesno/0_0_0_1_
↪0_1_1_0.wav"}], "sampling_rate": 8000, "num_samples": 48880,
↪"duration": 6.11, "channel_ids": [0]}
{"id": "0_0_1_0_0_1_1_0", "sources": [{"type": "file", "channels": [0],
↪ "source": "/tmp/icefall/egs/yesno/ASR/download/waves_yesno/0_0_1_0_
↪0_1_1_0.wav"}], "sampling_rate": 8000, "num_samples": 48160,
↪"duration": 6.02, "channel_ids": [0]}
```

Please refer to https://github.com/lhotse-speech/lhotse/blob/master/lhotse/audio.py#L300 for the meaning of each field per line.

```
gunzip -c  yesno_supervisions_train.jsonl.gz  | head -n 3
```

The output is given below:

```
{"id": "0_0_0_0_1_1_1_1", "recording_id": "0_0_0_0_1_1_1_1", "start": 0.0,
↪"duration": 6.35, "channel": 0, "text": "NO NO NO NO YES YES YES YES",
↪"language": "Hebrew"}
{"id": "0_0_0_1_0_1_1_0", "recording_id": "0_0_0_1_0_1_1_0", "start": 0.0,
↪"duration": 6.11, "channel": 0, "text": "NO NO NO YES NO YES YES NO",
↪"language": "Hebrew"}
{"id": "0_0_1_0_0_1_1_0", "recording_id": "0_0_1_0_0_1_1_0", "start": 0.0,
```

(continues on next page)

(continued from previous page)

```
↪"duration": 6.02, "channel": 0, "text": "NO NO YES NO NO YES YES NO",
↪"language": "Hebrew"}
```

Please refer to https://github.com/lhotse-speech/lhotse/blob/master/lhotse/supervision.py#L510 for the meaning of each field per line.

data/fbank:

This directory contains everything from `data/manifests`. Furthermore, it also contains features for training.

`data/fbank/yesno_feats_train.lca` contains the features for the train dataset. Features are compressed using lilcom.

`data/fbank/yesno_cuts_train.jsonl.gz` stores the CutSet, which stores RecordingSet, SupervisionSet, and FeatureSet.

To give you an idea about what it looks like, we can run the following command:

```
cd data/fbank

gunzip -c yesno_cuts_train.jsonl.gz | head -n 3
```

The output is given below:

```
{"id": "0_0_0_0_1_1_1_1-0", "start": 0, "duration": 6.35, "channel": 0,
↪ "supervisions": [{"id": "0_0_0_0_1_1_1_1", "recording_id": "0_0_0_0_
↪1_1_1_1", "start": 0.0, "duration": 6.35, "channel": 0, "text": "NO␣
↪NO NO NO YES YES YES YES", "language": "Hebrew"}], "features": {"type
↪": "kaldi-fbank", "num_frames": 635, "num_features": 23, "frame_shift
↪": 0.01, "sampling_rate": 8000, "start": 0, "duration": 6.35,
↪"storage_type": "lilcom_chunky", "storage_path": "data/fbank/yesno_
↪feats_train.lca", "storage_key": "0,13000,3570", "channels": 0},
↪"recording": {"id": "0_0_0_0_1_1_1_1", "sources": [{"type": "file",
↪"channels": [0], "source": "/tmp/icefall/egs/yesno/ASR/download/
↪waves_yesno/0_0_0_0_1_1_1_1.wav"}], "sampling_rate": 8000, "num_
↪samples": 50800, "duration": 6.35, "channel_ids": [0]}, "type":
↪"MonoCut"}
{"id": "0_0_0_1_0_1_1_0-1", "start": 0, "duration": 6.11, "channel": 0,
↪ "supervisions": [{"id": "0_0_0_1_0_1_1_0", "recording_id": "0_0_0_1_
↪0_1_1_0", "start": 0.0, "duration": 6.11, "channel": 0, "text": "NO␣
↪NO NO YES NO YES YES NO", "language": "Hebrew"}], "features": {"type
↪": "kaldi-fbank", "num_frames": 611, "num_features": 23, "frame_shift
↪": 0.01, "sampling_rate": 8000, "start": 0, "duration": 6.11,
↪"storage_type": "lilcom_chunky", "storage_path": "data/fbank/yesno_
↪feats_train.lca", "storage_key": "16570,12964,2929", "channels": 0},
↪"recording": {"id": "0_0_0_1_0_1_1_0", "sources": [{"type": "file",
↪"channels": [0], "source": "/tmp/icefall/egs/yesno/ASR/download/
↪waves_yesno/0_0_0_1_0_1_1_0.wav"}], "sampling_rate": 8000, "num_
↪samples": 48880, "duration": 6.11, "channel_ids": [0]}, "type":
↪"MonoCut"}
{"id": "0_0_1_0_0_1_1_0-2", "start": 0, "duration": 6.02, "channel": 0,
↪ "supervisions": [{"id": "0_0_1_0_0_1_1_0", "recording_id": "0_0_1_0_
↪0_1_1_0", "start": 0.0, "duration": 6.02, "channel": 0, "text": "NO␣
↪NO YES NO NO YES YES NO", "language": "Hebrew"}], "features": {"type
```

(continues on next page)

(continued from previous page)

```
↪": "kaldi-fbank", "num_frames": 602, "num_features": 23, "frame_shift
↪": 0.01, "sampling_rate": 8000, "start": 0, "duration": 6.02,
↪"storage_type": "lilcom_chunky", "storage_path": "data/fbank/yesno_
↪feats_train.lca", "storage_key": "32463,12936,2696", "channels": 0},
↪"recording": {"id": "0_0_1_0_0_1_1_0", "sources": [{"type": "file",
↪"channels": [0], "source": "/tmp/icefall/egs/yesno/ASR/download/
↪waves_yesno/0_0_1_0_0_1_1_0.wav"}], "sampling_rate": 8000, "num_
↪samples": 48160, "duration": 6.02, "channel_ids": [0]}, "type":
↪"MonoCut"}
```

Note that `yesno_cuts_train.jsonl.gz` only stores the information about how to read the features. The actual features are stored separately in `data/fbank/yesno_feats_train.lca`.

data/lang:

This directory contains the lexicon.

data/lm:

This directory contains language models.

1.3 Training

After *Data Preparation*, we can start training.

The command to start the training is quite simple:

```
cd /tmp/icefall
export PYTHONPATH=/tmp/icefall:$PYTHONPATH
cd egs/yesno/ASR

# We use CPU for training by setting the following environment variable
export CUDA_VISIBLE_DEVICES=""

./tdnn/train.py
```

That's it!

You can find the training logs below:

1.3.1 For the more curious

```
./tdnn/train.py --help
```

will print the usage information about `./tdnn/train.py`. For instance, you can specify the number of epochs to train and the location to save the training results.

The training text logs are saved in `tdnn/exp/log` while the tensorboard logs are in `tdnn/exp/tensorboard`.

1.4 Decoding

After *Training*, we can start decoding.

The command to start the decoding is quite simple:

```
cd /tmp/icefall
export PYTHONPATH=/tmp/icefall:$PYTHONPATH
cd egs/yesno/ASR

# We use CPU for decoding by setting the following environment variable
export CUDA_VISIBLE_DEVICES=""

./tdnn/decode.py
```

The output logs are given below:

1.4.1 For the more curious

```
./tdnn/decode.py --help
```

will print the usage information about `./tdnn/decode.py`. For instance, you can specify:

- `--epoch` to use which checkpoint for decoding
- `--avg` to select how many checkpoints to use for model averaging

You usually try different combinations of `--epoch` and `--avg` and select one that leads to the lowest WER (Word Error Rate).

1.5 Model Export

There are three ways to export a pre-trained model.

- Export the model parameters via model.state_dict()
- Export via torchscript: either torch.jit.script() or torch.jit.trace()
- Export to ONNX via torch.onnx.export()

Each method is explained below in detail.

1.5.1 Export the model parameters via model.state_dict()

The command for this kind of export is

```
cd /tmp/icefall
export PYTHONPATH=/tmp/icefall:$PYTHONPATH
cd egs/yesno/ASR

# assume that "--epoch 14 --avg 2" produces the lowest WER.

./tdnn/export.py --epoch 14 --avg 2
```

The output logs are given below:

```
2023-08-16 20:42:03,912 INFO [export.py:76] {'exp_dir': PosixPath('tdnn/exp'), 'lang_dir
↪': PosixPath('data/lang_phone'), 'lr': 0.01, 'feature_dim': 23, 'weight_decay': 1e-06,
↪'start_epoch': 0, 'best_train_loss': inf, 'best_valid_loss': inf, 'best_train_epoch': -
↪1, 'best_valid_epoch': -1, 'batch_idx_train': 0, 'log_interval': 10, 'reset_interval':␣
↪20, 'valid_interval': 10, 'beam_size': 10, 'reduction': 'sum', 'use_double_scores':␣
↪True, 'epoch': 14, 'avg': 2, 'jit': False}
2023-08-16 20:42:03,913 INFO [lexicon.py:168] Loading pre-compiled data/lang_phone/Linv.
↪pt
2023-08-16 20:42:03,950 INFO [export.py:93] averaging ['tdnn/exp/epoch-13.pt', 'tdnn/exp/
↪epoch-14.pt']
2023-08-16 20:42:03,971 INFO [export.py:106] Not using torch.jit.script
2023-08-16 20:42:03,974 INFO [export.py:111] Saved to tdnn/exp/pretrained.pt
```

We can see from the logs that the exported model is saved to the file `tdnn/exp/pretrained.pt`.

To give you an idea of what `tdnn/exp/pretrained.pt` contains, we can use the following command:

```
>>> import torch
>>> m = torch.load("tdnn/exp/pretrained.pt")
>>> list(m.keys())
['model']
>>> list(m["model"].keys())
['tdnn.0.weight', 'tdnn.0.bias', 'tdnn.2.running_mean', 'tdnn.2.running_var', 'tdnn.2.
↪num_batches_tracked', 'tdnn.3.weight', 'tdnn.3.bias', 'tdnn.5.running_mean', 'tdnn.5.
↪running_var', 'tdnn.5.num_batches_tracked', 'tdnn.6.weight', 'tdnn.6.bias', 'tdnn.8.
↪running_mean', 'tdnn.8.running_var', 'tdnn.8.num_batches_tracked', 'output_linear.
↪weight', 'output_linear.bias']
```

We can use `tdnn/exp/pretrained.pt` in the following way with `./tdnn/decode.py`:

```
cd tdnn/exp
ln -s pretrained.pt epoch-99.pt
cd ../..

./tdnn/decode.py --epoch 99 --avg 1
```

The output logs of the above command are given below:

```
2023-08-16 20:45:48,089 INFO [decode.py:262] Decoding started
2023-08-16 20:45:48,090 INFO [decode.py:263] {'exp_dir': PosixPath('tdnn/exp'), 'lang_dir
↪': PosixPath('data/lang_phone'), 'feature_dim': 23, 'search_beam': 20, 'output_beam':␣
↪8, 'min_active_states': 30, 'max_active_states': 10000, 'use_double_scores': True,
↪'epoch': 99, 'avg': 1, 'export': False, 'feature_dir': PosixPath('data/fbank'), 'max_
↪duration': 30.0, 'bucketing_sampler': False, 'num_buckets': 10, 'concatenate_cuts':␣
↪False, 'duration_factor': 1.0, 'gap': 1.0, 'on_the_fly_feats': False, 'shuffle': False,
↪ 'return_cuts': True, 'num_workers': 2, 'env_info': {'k2-version': '1.24.3', 'k2-build-
↪type': 'Release', 'k2-with-cuda': False, 'k2-git-sha1':
↪'ad79f1c699c684de9785ed6ca5edb805a41f78c3', 'k2-git-date': 'Wed Jul 26 09:30:42 2023',
↪'lhotse-version': '1.16.0.dev+git.aa073f6.clean', 'torch-version': '2.0.0', 'torch-
↪cuda-available': False, 'torch-cuda-version': None, 'python-version': '3.1', 'icefall-
↪git-branch': 'master', 'icefall-git-sha1': '9a47c08-clean', 'icefall-git-date': 'Mon␣
↪Aug 14 22:10:50 2023', 'icefall-path': '/private/tmp/icefall', 'k2-path': '/private/
↪tmp/icefall_env/lib/python3.11/site-packages/k2/__init__.py', 'lhotse-path': '/private/
```

(continues on next page)

(continued from previous page)

```
↪tmp/icefall_env/lib/python3.11/site-packages/lhotse/__init__.py', 'hostname':
↪'fangjuns-MacBook-Pro.local', 'IP address': '127.0.0.1'}}
2023-08-16 20:45:48,092 INFO [lexicon.py:168] Loading pre-compiled data/lang_phone/Linv.
↪pt
2023-08-16 20:45:48,103 INFO [decode.py:272] device: cpu
2023-08-16 20:45:48,109 INFO [checkpoint.py:112] Loading checkpoint from tdnn/exp/epoch-
↪99.pt
2023-08-16 20:45:48,115 INFO [asr_datamodule.py:218] About to get test cuts
2023-08-16 20:45:48,115 INFO [asr_datamodule.py:253] About to get test cuts
2023-08-16 20:45:50,386 INFO [decode.py:203] batch 0/?, cuts processed until now is 4
2023-08-16 20:45:50,556 INFO [decode.py:240] The transcripts are stored in tdnn/exp/
↪recogs-test_set.txt
2023-08-16 20:45:50,557 INFO [utils.py:564] [test_set] %WER 0.42% [1 / 240, 0 ins, 1 del,
↪ 0 sub ]
2023-08-16 20:45:50,558 INFO [decode.py:248] Wrote detailed error stats to tdnn/exp/errs-
↪test_set.txt
2023-08-16 20:45:50,559 INFO [decode.py:315] Done!
```

We can see that it produces an identical WER as before.

We can also use it to decode files with the following command:

```
# ./tdnn/pretrained.py requires kaldifeat
#
# Please refer to https://csukuangfj.github.io/kaldifeat/installation/from_wheels.html
# for how to install kaldifeat

pip install kaldifeat==1.25.0.dev20230726+cpu.torch2.0.0 -f https://csukuangfj.github.io/
↪kaldifeat/cpu.html

./tdnn/pretrained.py \
  --checkpoint ./tdnn/exp/pretrained.pt \
  --HLG ./data/lang_phone/HLG.pt \
  --words-file ./data/lang_phone/words.txt \
  download/waves_yesno/0_0_0_1_0_0_0_1.wav \
  download/waves_yesno/0_0_1_0_0_0_1_0.wav
```

The output is given below:

```
2023-08-16 20:53:19,208 INFO [pretrained.py:136] {'feature_dim': 23, 'num_classes': 4,
↪'sample_rate': 8000, 'search_beam': 20, 'output_beam': 8, 'min_active_states': 30,
↪'max_active_states': 10000, 'use_double_scores': True, 'checkpoint': './tdnn/exp/
↪pretrained.pt', 'words_file': './data/lang_phone/words.txt', 'HLG': './data/lang_phone/
↪HLG.pt', 'sound_files': ['download/waves_yesno/0_0_0_1_0_0_0_1.wav', 'download/waves_
↪yesno/0_0_1_0_0_0_1_0.wav']}
2023-08-16 20:53:19,208 INFO [pretrained.py:142] device: cpu
2023-08-16 20:53:19,208 INFO [pretrained.py:144] Creating model
2023-08-16 20:53:19,212 INFO [pretrained.py:156] Loading HLG from ./data/lang_phone/HLG.
↪pt
2023-08-16 20:53:19,213 INFO [pretrained.py:160] Constructing Fbank computer
2023-08-16 20:53:19,213 INFO [pretrained.py:170] Reading sound files: ['download/waves_
↪yesno/0_0_0_1_0_0_0_1.wav', 'download/waves_yesno/0_0_1_0_0_0_1_0.wav']
2023-08-16 20:53:19,224 INFO [pretrained.py:176] Decoding started
```

(continues on next page)

(continued from previous page)

```
2023-08-16 20:53:19,304 INFO [pretrained.py:212]
download/waves_yesno/0_0_0_1_0_0_0_1.wav:
NO NO NO YES NO NO NO YES

download/waves_yesno/0_0_1_0_0_0_1_0.wav:
NO NO YES NO NO NO YES NO


2023-08-16 20:53:19,304 INFO [pretrained.py:214] Decoding Done
```

1.5.2 Export via torch.jit.script()

The command for this kind of export is

```
cd /tmp/icefall
export PYTHONPATH=/tmp/icefall:$PYTHONPATH
cd egs/yesno/ASR

# assume that "--epoch 14 --avg 2" produces the lowest WER.

./tdnn/export.py --epoch 14 --avg 2 --jit true
```

The output logs are given below:

```
2023-08-16 20:47:44,666 INFO [export.py:76] {'exp_dir': PosixPath('tdnn/exp'), 'lang_dir
↪': PosixPath('data/lang_phone'), 'lr': 0.01, 'feature_dim': 23, 'weight_decay': 1e-06,
↪'start_epoch': 0, 'best_train_loss': inf, 'best_valid_loss': inf, 'best_train_epoch': -
↪1, 'best_valid_epoch': -1, 'batch_idx_train': 0, 'log_interval': 10, 'reset_interval':␣
↪20, 'valid_interval': 10, 'beam_size': 10, 'reduction': 'sum', 'use_double_scores':␣
↪True, 'epoch': 14, 'avg': 2, 'jit': True}
2023-08-16 20:47:44,667 INFO [lexicon.py:168] Loading pre-compiled data/lang_phone/Linv.
↪pt
2023-08-16 20:47:44,670 INFO [export.py:93] averaging ['tdnn/exp/epoch-13.pt', 'tdnn/exp/
↪epoch-14.pt']
2023-08-16 20:47:44,677 INFO [export.py:100] Using torch.jit.script
2023-08-16 20:47:44,843 INFO [export.py:104] Saved to tdnn/exp/cpu_jit.pt
```

From the output logs we can see that the generated file is saved to `tdnn/exp/cpu_jit.pt`.

Don't be confused by the name `cpu_jit.pt`. The `cpu` part means the model is moved to CPU before exporting. That means, when you load it with:

```
torch.jit.load()
```

you don't need to specify the argument map_location and it resides on CPU by default.

To use `tdnn/exp/cpu_jit.pt` with icefall to decode files, we can use:

```
# ./tdnn/jit_pretrained.py requires kaldifeat
#
# Please refer to https://csukuangfj.github.io/kaldifeat/installation/from_wheels.html
# for how to install kaldifeat
```

(continues on next page)

(continued from previous page)

```
pip install kaldifeat==1.25.0.dev20230726+cpu.torch2.0.0 -f https://csukuangfj.github.io/
↪kaldifeat/cpu.html


./tdnn/jit_pretrained.py \
  --nn-model ./tdnn/exp/cpu_jit.pt \
  --HLG ./data/lang_phone/HLG.pt \
  --words-file ./data/lang_phone/words.txt \
  download/waves_yesno/0_0_0_1_0_0_0_1.wav \
  download/waves_yesno/0_0_1_0_0_0_1_0.wav
```

The output is given below:

```
2023-08-16 20:56:00,603 INFO [jit_pretrained.py:121] {'feature_dim': 23, 'num_classes':␣
↪4, 'sample_rate': 8000, 'search_beam': 20, 'output_beam': 8, 'min_active_states': 30,
↪'max_active_states': 10000, 'use_double_scores': True, 'nn_model': './tdnn/exp/cpu_jit.
↪pt', 'words_file': './data/lang_phone/words.txt', 'HLG': './data/lang_phone/HLG.pt',
↪'sound_files': ['download/waves_yesno/0_0_0_1_0_0_0_1.wav', 'download/waves_yesno/0_0_
↪1_0_0_0_1_0.wav']}
2023-08-16 20:56:00,603 INFO [jit_pretrained.py:127] device: cpu
2023-08-16 20:56:00,603 INFO [jit_pretrained.py:129] Loading torchscript model
2023-08-16 20:56:00,640 INFO [jit_pretrained.py:134] Loading HLG from ./data/lang_phone/
↪HLG.pt
2023-08-16 20:56:00,641 INFO [jit_pretrained.py:138] Constructing Fbank computer
2023-08-16 20:56:00,641 INFO [jit_pretrained.py:148] Reading sound files: ['download/
↪waves_yesno/0_0_0_1_0_0_0_1.wav', 'download/waves_yesno/0_0_1_0_0_0_1_0.wav']
2023-08-16 20:56:00,642 INFO [jit_pretrained.py:154] Decoding started
2023-08-16 20:56:00,727 INFO [jit_pretrained.py:190]
download/waves_yesno/0_0_0_1_0_0_0_1.wav:
NO NO NO YES NO NO NO YES

download/waves_yesno/0_0_1_0_0_0_1_0.wav:
NO NO YES NO NO NO YES NO


2023-08-16 20:56:00,727 INFO [jit_pretrained.py:192] Decoding Done
```

Hint: We provide only code for `torch.jit.script()`. You can try `torch.jit.trace()` if you want.

1.5.3 Export via torch.onnx.export()

The command for this kind of export is

```
cd /tmp/icefall
export PYTHONPATH=/tmp/icefall:$PYTHONPATH
cd egs/yesno/ASR

# tdnn/export_onnx.py requires onnx and onnxruntime
```

(continues on next page)

(continued from previous page)

```
pip install onnx onnxruntime

# assume that "--epoch 14 --avg 2" produces the lowest WER.

./tdnn/export_onnx.py \
  --epoch 14 \
  --avg 2
```

The output logs are given below:

```
2023-08-16 20:59:20,888 INFO [export_onnx.py:83] {'exp_dir': PosixPath('tdnn/exp'),
↪'lang_dir': PosixPath('data/lang_phone'), 'lr': 0.01, 'feature_dim': 23, 'weight_decay
↪': 1e-06, 'start_epoch': 0, 'best_train_loss': inf, 'best_valid_loss': inf, 'best_
↪train_epoch': -1, 'best_valid_epoch': -1, 'batch_idx_train': 0, 'log_interval': 10,
↪'reset_interval': 20, 'valid_interval': 10, 'beam_size': 10, 'reduction': 'sum', 'use_
↪double_scores': True, 'epoch': 14, 'avg': 2}
2023-08-16 20:59:20,888 INFO [lexicon.py:168] Loading pre-compiled data/lang_phone/Linv.
↪pt
2023-08-16 20:59:20,892 INFO [export_onnx.py:100] averaging ['tdnn/exp/epoch-13.pt',
↪'tdnn/exp/epoch-14.pt']
================ Diagnostic Run torch.onnx.export version 2.0.0 ================
verbose: False, log level: Level.ERROR
======================= 0 NONE 0 NOTE 0 WARNING 0 ERROR ========================

2023-08-16 20:59:21,047 INFO [export_onnx.py:127] Saved to tdnn/exp/model-epoch-14-avg-2.
↪onnx
2023-08-16 20:59:21,047 INFO [export_onnx.py:136] meta_data: {'model_type': 'tdnn',
↪'version': '1', 'model_author': 'k2-fsa', 'comment': 'non-streaming tdnn for the yesno␣
↪recipe', 'vocab_size': 4}
2023-08-16 20:59:21,049 INFO [export_onnx.py:140] Generate int8 quantization models
2023-08-16 20:59:21,075 INFO [onnx_quantizer.py:538] Quantization parameters for tensor:
↪"/Transpose_1_output_0" not specified
2023-08-16 20:59:21,081 INFO [export_onnx.py:151] Saved to tdnn/exp/model-epoch-14-avg-2.
↪int8.onnx
```

We can see from the logs that it generates two files:

- `tdnn/exp/model-epoch-14-avg-2.onnx` (ONNX model with `float32` weights)
- `tdnn/exp/model-epoch-14-avg-2.int8.onnx` (ONNX model with `int8` weights)

To use the generated ONNX model files for decoding with onnxruntime, we can use

```
# ./tdnn/onnx_pretrained.py requires kaldifeat
#
# Please refer to https://csukuangfj.github.io/kaldifeat/installation/from_wheels.html
# for how to install kaldifeat

pip install kaldifeat==1.25.0.dev20230726+cpu.torch2.0.0 -f https://csukuangfj.github.io/
↪kaldifeat/cpu.html

./tdnn/onnx_pretrained.py \
  --nn-model ./tdnn/exp/model-epoch-14-avg-2.onnx \
  --HLG ./data/lang_phone/HLG.pt \
```

(continues on next page)

(continued from previous page)

```
  --words-file ./data/lang_phone/words.txt \
  download/waves_yesno/0_0_0_1_0_0_0_1.wav \
  download/waves_yesno/0_0_1_0_0_0_1_0.wav
```

The output is given below:

```
2023-08-16 21:03:24,260 INFO [onnx_pretrained.py:166] {'feature_dim': 23, 'sample_rate':␣
↪8000, 'search_beam': 20, 'output_beam': 8, 'min_active_states': 30, 'max_active_states
↪': 10000, 'use_double_scores': True, 'nn_model': './tdnn/exp/model-epoch-14-avg-2.onnx
↪', 'words_file': './data/lang_phone/words.txt', 'HLG': './data/lang_phone/HLG.pt',
↪'sound_files': ['download/waves_yesno/0_0_0_1_0_0_0_1.wav', 'download/waves_yesno/0_0_
↪1_0_0_0_1_0.wav']}
2023-08-16 21:03:24,260 INFO [onnx_pretrained.py:171] device: cpu
2023-08-16 21:03:24,260 INFO [onnx_pretrained.py:173] Loading onnx model ./tdnn/exp/
↪model-epoch-14-avg-2.onnx
2023-08-16 21:03:24,267 INFO [onnx_pretrained.py:176] Loading HLG from ./data/lang_phone/
↪HLG.pt
2023-08-16 21:03:24,270 INFO [onnx_pretrained.py:180] Constructing Fbank computer
2023-08-16 21:03:24,273 INFO [onnx_pretrained.py:190] Reading sound files: ['download/
↪waves_yesno/0_0_0_1_0_0_0_1.wav', 'download/waves_yesno/0_0_1_0_0_0_1_0.wav']
2023-08-16 21:03:24,279 INFO [onnx_pretrained.py:196] Decoding started
2023-08-16 21:03:24,318 INFO [onnx_pretrained.py:232]
download/waves_yesno/0_0_0_1_0_0_0_1.wav:
NO NO NO YES NO NO NO YES

download/waves_yesno/0_0_1_0_0_0_1_0.wav:
NO NO YES NO NO NO YES NO


2023-08-16 21:03:24,318 INFO [onnx_pretrained.py:234] Decoding Done
```

Note: To use the `int8` ONNX model for decoding, please use:

```
./tdnn/onnx_pretrained.py \
  --nn-model ./tdnn/exp/model-epoch-14-avg-2.onnx \
  --HLG ./data/lang_phone/HLG.pt \
  --words-file ./data/lang_phone/words.txt \
  download/waves_yesno/0_0_0_1_0_0_0_1.wav \
  download/waves_yesno/0_0_1_0_0_0_1_0.wav
```

1.5.4 For the more curious

If you are wondering how to deploy the model without `torch`, please continue reading. We will show how to use sherpa-onnx to run the exported ONNX models, which depends only on onnxruntime and does not depend on `torch`.

In this tutorial, we will only demonstrate the usage of sherpa-onnx with the pre-trained model of the yesno recipe. There are also other two frameworks available:

- **`sherpa`**_. It works with torchscript models.
- sherpa-ncnn. It works with models exported using icefall_export_to_ncnn with ncnn

Please see https://k2-fsa.github.io/sherpa/ for further details.

CHAPTER

TWO

INSTALLATION

Hint: We also provide *Docker* support, which has already setup the environment for you.

Hint: We have a colab notebook guiding you step by step to setup the environment.

icefall depends on k2 and lhotse.

We recommend that you use the following steps to install the dependencies.

- (0) Install CUDA toolkit and cuDNN
- (1) Install torch and torchaudio
- (2) Install k2
- (3) Install lhotse

Caution: Installation order matters.

2.1 (0) Install CUDA toolkit and cuDNN

Please refer to https://k2-fsa.github.io/k2/installation/cuda-cudnn.html to install CUDA and cuDNN.

2.2 (1) Install torch and torchaudio

Please refer https://pytorch.org/ to install torch and torchaudio.

Caution: Please install torch and torchaudio at the same time.

2.3 (2) Install k2

Please refer to https://k2-fsa.github.io/k2/installation/index.html to install k2.

Caution: Please don't change your installed PyTorch after you have installed k2.

Note: We suggest that you install k2 from pre-compiled wheels by following https://k2-fsa.github.io/k2/installation/from_wheels.html

Hint: Please always install the latest version of k2.

2.4 (3) Install lhotse

Please refer to https://lhotse.readthedocs.io/en/latest/getting-started.html#installation to install lhotse.

Hint: We strongly recommend you to use:

```
pip install git+https://github.com/lhotse-speech/lhotse
```

to install the latest version of lhotse.

2.5 (4) Download icefall

icefall is a collection of Python scripts; what you need is to download it and set the environment variable PYTHONPATH to point to it.

Assume you want to place icefall in the folder /tmp. The following commands show you how to setup icefall:

```
cd /tmp
git clone https://github.com/k2-fsa/icefall
cd icefall
pip install -r requirements.txt
export PYTHONPATH=/tmp/icefall:$PYTHONPATH
```

Hint: You can put several versions of icefall in the same virtual environment. To switch among different versions of icefall, just set PYTHONPATH to point to the version you want.

2.6 Installation example

The following shows an example about setting up the environment.

2.6.1 (1) Create a virtual environment

```
kuangfangjun:~$ virtualenv -p python3.8 test-icefall
created virtual environment CPython3.8.0.final.0-64 in 9422ms
  creator CPython3Posix(dest=/star-fj/fangjun/test-icefall, clear=False, no_vcs_
↪ignore=False, global=False)
  seeder FromAppData(download=False, pip=bundle, setuptools=bundle, wheel=bundle,␣
↪via=copy, app_data_dir=/star-fj/fangjun/.local/share/virtualenv)
    added seed packages: pip==22.3.1, setuptools==65.6.3, wheel==0.38.4
  activators BashActivator,CShellActivator,FishActivator,NushellActivator,
↪PowerShellActivator,PythonActivator

kuangfangjun:~$ source test-icefall/bin/activate

(test-icefall) kuangfangjun:~$
```

2.6.2 (2) Install CUDA toolkit and cuDNN

You need to determine the version of CUDA toolkit to install.

```
(test-icefall) kuangfangjun:~$ nvidia-smi | head -n 4

Wed Jul 26 21:57:49 2023
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 510.47.03    Driver Version: 510.47.03    CUDA Version: 11.6     |
|-------------------------------+----------------------+----------------------+
```

You can choose any CUDA version that is `not` greater than the version printed by `nvidia-smi`. In our case, we can choose any version `<= 11.6`.

We will use `CUDA 11.6` in this example. Please follow https://k2-fsa.github.io/k2/installation/cuda-cudnn.html#cuda-11-6 to install CUDA toolkit and cuDNN if you have not done that before.

After installing CUDA toolkit, you can use the following command to verify it:

```
(test-icefall) kuangfangjun:~$ nvcc --version

nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2019 NVIDIA Corporation
Built on Wed_Oct_23_19:24:38_PDT_2019
Cuda compilation tools, release 10.2, V10.2.89
```

2.6.3 (3) Install torch and torchaudio

Since we have selected CUDA toolkit `11.6`, we have to install a version of torch that is compiled against CUDA `11.6`. We select `torch 1.13.0+cu116` in this example.

After selecting the version of torch to install, we need to also install a compatible version of torchaudio, which is `0.13.0+cu116` in our case.

Please refer to https://pytorch.org/audio/stable/installation.html#compatibility-matrix to select an appropriate version of torchaudio to install if you use a different version of torch.

```
(test-icefall) kuangfangjun:~$ pip install torch==1.13.0+cu116 torchaudio==0.13.0+cu116 -
↪f https://download.pytorch.org/whl/torch_stable.html

Looking in links: https://download.pytorch.org/whl/torch_stable.html
Collecting torch==1.13.0+cu116
  Downloading https://download.pytorch.org/whl/cu116/torch-1.13.0%2Bcu116-cp38-cp38-
↪linux_x86_64.whl (1983.0 MB)
     ________________________________________ 2.0/2.0 GB 764.4 kB/s eta 0:00:00
Collecting torchaudio==0.13.0+cu116
  Downloading https://download.pytorch.org/whl/cu116/torchaudio-0.13.0%2Bcu116-cp38-cp38-
↪linux_x86_64.whl (4.2 MB)
     ________________________________________ 4.2/4.2 MB 1.3 MB/s eta 0:00:00
Requirement already satisfied: typing-extensions in /star-fj/fangjun/test-icefall/lib/
↪python3.8/site-packages (from torch==1.13.0+cu116) (4.7.1)
Installing collected packages: torch, torchaudio
Successfully installed torch-1.13.0+cu116 torchaudio-0.13.0+cu116
```

Verify that torch and torchaudio are successfully installed:

```
(test-icefall) kuangfangjun:~$ python3 -c "import torch; print(torch.__version__)"

1.13.0+cu116

(test-icefall) kuangfangjun:~$ python3 -c "import torchaudio; print(torchaudio.__version_
↪_)"

0.13.0+cu116
```

2.6.4 (4) Install k2

We will install k2 from pre-compiled wheels by following https://k2-fsa.github.io/k2/installation/from_wheels.html

```
(test-icefall) kuangfangjun:~$ pip install k2==1.24.3.dev20230725+cuda11.6.torch1.13.0 -
↪f https://k2-fsa.github.io/k2/cuda.html

Looking in indexes: https://pypi.tuna.tsinghua.edu.cn/simple
Looking in links: https://k2-fsa.github.io/k2/cuda.html
Collecting k2==1.24.3.dev20230725+cuda11.6.torch1.13.0
  Downloading https://huggingface.co/csukuangfj/k2/resolve/main/ubuntu-cuda/k2-1.24.3.
↪dev20230725%2Bcuda11.6.torch1.13.0-cp38-cp38-manylinux_2_17_x86_64.manylinux2014_x86_
↪64.whl (104.3 MB)
     ________________________________________ 104.3/104.3 MB 5.1 MB/s eta 0:00:00
```

(continues on next page)

(continued from previous page)

```
Requirement already satisfied: torch==1.13.0 in /star-fj/fangjun/test-icefall/lib/
↪python3.8/site-packages (from k2==1.24.3.dev20230725+cuda11.6.torch1.13.0) (1.13.
↪0+cu116)
Collecting graphviz
  Using cached https://pypi.tuna.tsinghua.edu.cn/packages/de/5e/
↪fcbb22c68208d39edff467809d06c9d81d7d27426460ebc598e55130c1aa/graphviz-0.20.1-py3-none-
↪any.whl (47 kB)
Requirement already satisfied: typing-extensions in /star-fj/fangjun/test-icefall/lib/
↪python3.8/site-packages (from torch==1.13.0->k2==1.24.3.dev20230725+cuda11.6.torch1.13.
↪0) (4.7.1)
Installing collected packages: graphviz, k2
Successfully installed graphviz-0.20.1 k2-1.24.3.dev20230725+cuda11.6.torch1.13.0
```

Hint: Please refer to https://k2-fsa.github.io/k2/cuda.html for the available pre-compiled wheels about k2.

Verify that k2 has been installed successfully:

```
(test-icefall) kuangfangjun:~$ python3 -m k2.version

Collecting environment information...

k2 version: 1.24.3
Build type: Release
Git SHA1: 4c05309499a08454997adf500b56dcc629e35ae5
Git date: Tue Jul 25 16:23:36 2023
Cuda used to build k2: 11.6
cuDNN used to build k2: 8.3.2
Python version used to build k2: 3.8
OS used to build k2: CentOS Linux release 7.9.2009 (Core)
CMake version: 3.27.0
GCC version: 9.3.1
CMAKE_CUDA_FLAGS:  -Wno-deprecated-gpu-targets   -lineinfo --expt-extended-lambda -use_
↪fast_math -Xptxas=-w  --expt-extended-lambda -gencode arch=compute_35,code=sm_35  -
↪lineinfo --expt-extended-lambda -use_fast_math -Xptxas=-w  --expt-extended-lambda -
↪gencode arch=compute_50,code=sm_50  -lineinfo --expt-extended-lambda -use_fast_math -
↪Xptxas=-w  --expt-extended-lambda -gencode arch=compute_60,code=sm_60  -lineinfo --
↪expt-extended-lambda -use_fast_math -Xptxas=-w  --expt-extended-lambda -gencode␣
↪arch=compute_61,code=sm_61  -lineinfo --expt-extended-lambda -use_fast_math -Xptxas=-w␣
↪  --expt-extended-lambda -gencode arch=compute_70,code=sm_70  -lineinfo --expt-extended-
↪lambda -use_fast_math -Xptxas=-w  --expt-extended-lambda -gencode arch=compute_75,
↪code=sm_75  -lineinfo --expt-extended-lambda -use_fast_math -Xptxas=-w  --expt-
↪extended-lambda -gencode arch=compute_80,code=sm_80  -lineinfo --expt-extended-lambda -
↪use_fast_math -Xptxas=-w  --expt-extended-lambda -gencode arch=compute_86,code=sm_86 -
↪DONNX_NAMESPACE=onnx_c2 -gencode arch=compute_35,code=sm_35 -gencode arch=compute_50,
↪code=sm_50 -gencode arch=compute_52,code=sm_52 -gencode arch=compute_60,code=sm_60 -
↪gencode arch=compute_61,code=sm_61 -gencode arch=compute_70,code=sm_70 -gencode␣
↪arch=compute_75,code=sm_75 -gencode arch=compute_80,code=sm_80 -gencode arch=compute_
↪86,code=sm_86 -gencode arch=compute_86,code=compute_86 -Xcudafe --diag_suppress=cc_
↪clobber_ignored,--diag_suppress=integer_sign_change,--diag_suppress=useless_using_
↪declaration,--diag_suppress=set_but_not_used,--diag_suppress=field_without_dll_
↪interface,--diag_suppress=base_class_has_different_dll_interface,--diag_suppress=dll_
```

(continues on next page)

(continued from previous page)

```
↪interface_conflict_none_assumed,--diag_suppress=dll_interface_conflict_dllexport_
↪assumed,--diag_suppress=implicit_return_from_non_void_function,--diag_
↪suppress=unsigned_compare_with_zero,--diag_suppress=declared_but_not_referenced,--diag_
↪suppress=bad_friend_decl --expt-relaxed-constexpr --expt-extended-lambda -D_GLIBCXX_
↪USE_CXX11_ABI=0 --compiler-options -Wall  --compiler-options -Wno-strict-overflow  --
↪compiler-options -Wno-unknown-pragmas
CMAKE_CXX_FLAGS:  -D_GLIBCXX_USE_CXX11_ABI=0 -Wno-unused-variable  -Wno-strict-overflow
PyTorch version used to build k2: 1.13.0+cu116
PyTorch is using Cuda: 11.6
NVTX enabled: True
With CUDA: True
Disable debug: True
Sync kernels : False
Disable checks: False
Max cpu memory allocate: 214748364800 bytes (or 200.0 GB)
k2 abort: False
__file__: /star-fj/fangjun/test-icefall/lib/python3.8/site-packages/k2/version/version.py
_k2.__file__: /star-fj/fangjun/test-icefall/lib/python3.8/site-packages/_k2.cpython-38-
↪x86_64-linux-gnu.so
```

2.6.5 (5) Install lhotse

```
(test-icefall) kuangfangjun:~$ pip install git+https://github.com/lhotse-speech/lhotse

Collecting git+https://github.com/lhotse-speech/lhotse
  Cloning https://github.com/lhotse-speech/lhotse to /tmp/pip-req-build-vq12fd5i
  Running command git clone --filter=blob:none --quiet https://github.com/lhotse-speech/
↪lhotse /tmp/pip-req-build-vq12fd5i
  Resolved https://github.com/lhotse-speech/lhotse to commit␣
↪7640d663469b22cd0b36f3246ee9b849cd25e3b7
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Preparing metadata (pyproject.toml) ... done
Collecting cytoolz>=0.10.1
  Downloading https://pypi.tuna.tsinghua.edu.cn/packages/1e/3b/
↪a7828d575aa17fb7acaf1ced49a3655aa36dad7e16eb7e6a2e4df0dda76f/cytoolz-0.12.2-cp38-cp38-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl (2.0 MB)
     ________________________________________ 2.0/2.0 MB 33.2 MB/s eta 0:00:00
Collecting pyyaml>=5.3.1
  Downloading https://pypi.tuna.tsinghua.edu.cn/packages/c8/6b/
↪6600ac24725c7388255b2f5add93f91e58a5d7efaf4af244fdbcc11a541b/PyYAML-6.0.1-cp38-cp38-ma
nylinux_2_17_x86_64.manylinux2014_x86_64.whl (736 kB)
     ________________________________________ 736.6/736.6 kB 38.6 MB/s eta 0:00:00
Collecting dataclasses
  Downloading https://pypi.tuna.tsinghua.edu.cn/packages/26/2f/
↪1095cdc2868052dd1e64520f7c0d5c8c550ad297e944e641dbf1ffbb9a5d/dataclasses-0.6-py3-none-
any.whl (14 kB)
Requirement already satisfied: torchaudio in ./test-icefall/lib/python3.8/site-packages␣
↪(from lhotse==1.16.0.dev0+git.7640d66.clean) (0.13.0+cu116)
Collecting lilcom>=1.1.0
```

(continues on next page)

(continued from previous page)

```
  Downloading https://pypi.tuna.tsinghua.edu.cn/packages/a8/65/
↪df0a69c52bd085ca1ad4e5c4c1a5c680e25f9477d8e49316c4ff1e5084a4/lilcom-1.7-cp38-cp38-many
linux_2_17_x86_64.manylinux2014_x86_64.whl (87 kB)
     ________________________________________ 87.1/87.1 kB 8.7 MB/s eta 0:00:00
Collecting tqdm
  Using cached https://pypi.tuna.tsinghua.edu.cn/packages/e6/02/
↪a2cff6306177ae6bc73bc0665065de51dfb3b9db7373e122e2735faf0d97/tqdm-4.65.0-py3-none-any
.whl (77 kB)
Requirement already satisfied: numpy>=1.18.1 in ./test-icefall/lib/python3.8/site-
↪packages (from lhotse==1.16.0.dev0+git.7640d66.clean) (1.24.4)
Collecting audioread>=2.1.9
  Using cached https://pypi.tuna.tsinghua.edu.cn/packages/5d/cb/
↪82a002441902dccbe427406785db07af10182245ee639ea9f4d92907c923/audioread-3.0.0.tar.gz (
377 kB)
  Preparing metadata (setup.py) ... done
Collecting tabulate>=0.8.1
  Using cached https://pypi.tuna.tsinghua.edu.cn/packages/40/44/
↪4a5f08c96eb108af5cb50b41f76142f0afa346dfa99d5296fe7202a11854/tabulate-0.9.0-py3-none-
any.whl (35 kB)
Collecting click>=7.1.1
  Downloading https://pypi.tuna.tsinghua.edu.cn/packages/1a/70/
↪e63223f8116931d365993d4a6b7ef653a4d920b41d03de7c59499962821f/click-8.1.6-py3-none-any.
whl (97 kB)
     ________________________________________ 97.9/97.9 kB 8.4 MB/s eta 0:00:00
Collecting packaging
  Using cached https://pypi.tuna.tsinghua.edu.cn/packages/ab/c3/
↪57f0601a2d4fe15de7a553c00adbc901425661bf048f2a22dfc500caf121/packaging-23.1-py3-none-
any.whl (48 kB)
Collecting intervaltree>=3.1.0
  Downloading https://pypi.tuna.tsinghua.edu.cn/packages/50/fb/
↪396d568039d21344639db96d940d40eb62befe704ef849b27949ded5c3bb/intervaltree-3.1.0.tar.gz
 (32 kB)
  Preparing metadata (setup.py) ... done
Requirement already satisfied: torch in ./test-icefall/lib/python3.8/site-packages (from␣
↪lhotse==1.16.0.dev0+git.7640d66.clean) (1.13.0+cu116)
Collecting SoundFile>=0.10
  Downloading https://pypi.tuna.tsinghua.edu.cn/packages/ad/bd/
↪0602167a213d9184fc688b1086dc6d374b7ae8c33eccf169f9b50ce6568c/soundfile-0.12.1-py2.py3-
none-manylinux_2_17_x86_64.whl (1.3 MB)
     ________________________________________ 1.3/1.3 MB 46.5 MB/s eta 0:00:00
Collecting toolz>=0.8.0
  Using cached https://pypi.tuna.tsinghua.edu.cn/packages/7f/5c/
↪922a3508f5bda2892be3df86c74f9cf1e01217c2b1f8a0ac4841d903e3e9/toolz-0.12.0-py3-none-any.
↪whl (55 kB)
Collecting sortedcontainers<3.0,>=2.0
  Using cached https://pypi.tuna.tsinghua.edu.cn/packages/32/46/
↪9cb0e58b2deb7f82b84065f37f3bffeb12413f947f9388e4cac22c4621ce/sortedcontainers-2.4.0-
↪py2.py3-none-any.whl (29 kB)
Collecting cffi>=1.0
  Using cached https://pypi.tuna.tsinghua.edu.cn/packages/b7/8b/
↪06f30caa03b5b3ac006de4f93478dbd0239e2a16566d81a106c322dc4f79/cffi-1.15.1-cp38-cp38-
↪manylinux_2_17_x86_64.manylinux2014_x86_64.whl (442 kB)
```

(continues on next page)

(continued from previous page)

```
Requirement already satisfied: typing-extensions in ./test-icefall/lib/python3.8/site-
↪packages (from torch->lhotse==1.16.0.dev0+git.7640d66.clean) (4.7.1)
Collecting pycparser
  Using cached https://pypi.tuna.tsinghua.edu.cn/packages/62/d5/
↪5f610ebe421e85889f2e55e33b7f9a6795bd982198517d912eb1c76e1a53/pycparser-2.21-py2.py3-
↪none-any.whl (118 kB)
Building wheels for collected packages: lhotse, audioread, intervaltree
  Building wheel for lhotse (pyproject.toml) ... done
  Created wheel for lhotse: filename=lhotse-1.16.0.dev0+git.7640d66.clean-py3-none-any.
↪whl size=687627 sha256=cbf0a4d2d0b639b33b91637a4175bc251d6a021a069644ecb1a9f2b3a83d072a
  Stored in directory: /tmp/pip-ephem-wheel-cache-wwtk90_m/wheels/7f/7a/8e/
↪a0bf241336e2e3cb573e1e21e5600952d49f5162454f2e612f
  Building wheel for audioread (setup.py) ... done
  Created wheel for audioread: filename=audioread-3.0.0-py3-none-any.whl size=23704␣
↪sha256=5e2d3537c96ce9cf0f645a654c671163707bf8cb8d9e358d0e2b0939a85ff4c2
  Stored in directory: /star-fj/fangjun/.cache/pip/wheels/e2/c3/9c/
↪f19ae5a03f8862d9f0776b0c0570f1fdd60a119d90954e3f39
  Building wheel for intervaltree (setup.py) ... done
  Created wheel for intervaltree: filename=intervaltree-3.1.0-py2.py3-none-any.whl␣
↪size=26098 sha256=2604170976cfffe0d2f678cb1a6e5b525f561cd50babe53d631a186734fec9f9
  Stored in directory: /star-fj/fangjun/.cache/pip/wheels/f3/ed/2b/
↪c179ebfad4e15452d6baef59737f27beb9bfb442e0620f7271
Successfully built lhotse audioread intervaltree
Installing collected packages: sortedcontainers, dataclasses, tqdm, toolz, tabulate,␣
↪pyyaml, pycparser, packaging, lilcom, intervaltree, click, audioread, cytoolz, cffi,␣
↪SoundFile, lhotse
Successfully installed SoundFile-0.12.1 audioread-3.0.0 cffi-1.15.1 click-8.1.6 cytoolz-
↪0.12.2 dataclasses-0.6 intervaltree-3.1.0 lhotse-1.16.0.dev0+git.7640d66.clean lilcom-
↪1.7 packaging-23.1 pycparser-2.21 pyyaml-6.0.1 sortedcontainers-2.4.0 tabulate-0.9.0␣
↪toolz-0.12.0 tqdm-4.65.0
```

Verify that lhotse has been installed successfully:

```
(test-icefall) kuangfangjun:~$ python3 -c "import lhotse; print(lhotse.__version__)"

1.16.0.dev+git.7640d66.clean
```

2.6.6 (6) Download icefall

```
(test-icefall) kuangfangjun:~$ cd /tmp/

(test-icefall) kuangfangjun:tmp$ git clone https://github.com/k2-fsa/icefall

Cloning into 'icefall'...
remote: Enumerating objects: 12942, done.
remote: Counting objects: 100% (67/67), done.
remote: Compressing objects: 100% (56/56), done.
remote: Total 12942 (delta 17), reused 35 (delta 6), pack-reused 12875
Receiving objects: 100% (12942/12942), 14.77 MiB | 9.29 MiB/s, done.
Resolving deltas: 100% (8835/8835), done.
```

(continues on next page)

(continued from previous page)

```
(test-icefall) kuangfangjun:tmp$ cd icefall/

(test-icefall) kuangfangjun:icefall$ pip install -r ./requirements.txt
```

2.7 Test Your Installation

To test that your installation is successful, let us run the yesno recipe on CPU.

2.7.1 Data preparation

```
(test-icefall) kuangfangjun:icefall$ export PYTHONPATH=/tmp/icefall:$PYTHONPATH

(test-icefall) kuangfangjun:icefall$ cd /tmp/icefall

(test-icefall) kuangfangjun:icefall$ cd egs/yesno/ASR

(test-icefall) kuangfangjun:ASR$ ./prepare.sh
```

The log of running `./prepare.sh` is:

```
2023-07-27 12:41:39 (prepare.sh:27:main) dl_dir: /tmp/icefall/egs/yesno/ASR/download
2023-07-27 12:41:39 (prepare.sh:30:main) Stage 0: Download data
/tmp/icefall/egs/yesno/ASR/download/waves_yesno.tar.gz: 100%|_____________________________
↪________________________| 4.70M/4.70M [00:00<00:00, 11.1MB/s]
2023-07-27 12:41:46 (prepare.sh:39:main) Stage 1: Prepare yesno manifest
2023-07-27 12:41:50 (prepare.sh:45:main) Stage 2: Compute fbank for yesno
2023-07-27 12:41:55,718 INFO [compute_fbank_yesno.py:65] Processing train
Extracting and storing features: 100%|__________________________________________________
↪_____________________________| 90/90 [00:01<00:00, 87.82it/s]
2023-07-27 12:41:56,778 INFO [compute_fbank_yesno.py:65] Processing test
Extracting and storing features: 100%|__________________________________________________
↪____________________________| 30/30 [00:00<00:00, 256.92it/s]
2023-07-27 12:41:57 (prepare.sh:51:main) Stage 3: Prepare lang
2023-07-27 12:42:02 (prepare.sh:66:main) Stage 4: Prepare G
/project/kaldilm/csrc/arpa_file_parser.cc:void␣
↪kaldilm::ArpaFileParser::Read(std::istream&):79
[I] Reading \data\ section.
/project/kaldilm/csrc/arpa_file_parser.cc:void␣
↪kaldilm::ArpaFileParser::Read(std::istream&):140
[I] Reading \1-grams: section.
2023-07-27 12:42:02 (prepare.sh:92:main) Stage 5: Compile HLG
2023-07-27 12:42:07,275 INFO [compile_hlg.py:124] Processing data/lang_phone
2023-07-27 12:42:07,276 INFO [lexicon.py:171] Converting L.pt to Linv.pt
2023-07-27 12:42:07,309 INFO [compile_hlg.py:48] Building ctc_topo. max_token_id: 3
2023-07-27 12:42:07,310 INFO [compile_hlg.py:52] Loading G.fst.txt
2023-07-27 12:42:07,314 INFO [compile_hlg.py:62] Intersecting L and G
2023-07-27 12:42:07,323 INFO [compile_hlg.py:64] LG shape: (4, None)
2023-07-27 12:42:07,323 INFO [compile_hlg.py:66] Connecting LG
```

(continues on next page)

(continued from previous page)

```
2023-07-27 12:42:07,323 INFO [compile_hlg.py:68] LG shape after k2.connect: (4, None)
2023-07-27 12:42:07,323 INFO [compile_hlg.py:70] <class 'torch.Tensor'>
2023-07-27 12:42:07,323 INFO [compile_hlg.py:71] Determinizing LG
2023-07-27 12:42:07,341 INFO [compile_hlg.py:74] <class '_k2.ragged.RaggedTensor'>
2023-07-27 12:42:07,341 INFO [compile_hlg.py:76] Connecting LG after k2.determinize
2023-07-27 12:42:07,341 INFO [compile_hlg.py:79] Removing disambiguation symbols on LG
2023-07-27 12:42:07,354 INFO [compile_hlg.py:91] LG shape after k2.remove_epsilon: (6,␣
↪None)
2023-07-27 12:42:07,445 INFO [compile_hlg.py:96] Arc sorting LG
2023-07-27 12:42:07,445 INFO [compile_hlg.py:99] Composing H and LG
2023-07-27 12:42:07,446 INFO [compile_hlg.py:106] Connecting LG
2023-07-27 12:42:07,446 INFO [compile_hlg.py:109] Arc sorting LG
2023-07-27 12:42:07,447 INFO [compile_hlg.py:111] HLG.shape: (8, None)
2023-07-27 12:42:07,447 INFO [compile_hlg.py:127] Saving HLG.pt to data/lang_phone
```

2.7.2 Training

Now let us run the training part:

```
(test-icefall) kuangfangjun:ASR$ export CUDA_VISIBLE_DEVICES=""

(test-icefall) kuangfangjun:ASR$ ./tdnn/train.py
```

Caution: We use `export CUDA_VISIBLE_DEVICES=""` so that icefall uses CPU even if there are GPUs available.

Hint: In case you get a `Segmentation fault (core dump)` error, please use:

```
export PROTOCOL_BUFFERS_PYTHON_IMPLEMENTATION=python
```

See more at <*https://github.com/k2-fsa/icefall/issues/674*> if you are interested.

The training log is given below:

```
2023-07-27 12:50:51,936 INFO [train.py:481] Training started
2023-07-27 12:50:51,936 INFO [train.py:482] {'exp_dir': PosixPath('tdnn/exp'), 'lang_dir
↪': PosixPath('data/lang_phone'), 'lr': 0.01, 'feature_dim': 23, 'weight_decay': 1e-06,
↪'start_epoch': 0, 'best_train_loss': inf, 'best_valid_loss': inf, 'best_train_epoch': -
↪1, 'best_valid_epoch': -1, 'batch_idx_train': 0, 'log_interval': 10, 'reset_interval':␣
↪20, 'valid_interval': 10, 'beam_size': 10, 'reduction': 'sum', 'use_double_scores':␣
↪True, 'world_size': 1, 'master_port': 12354, 'tensorboard': True, 'num_epochs': 15,
↪'seed': 42, 'feature_dir': PosixPath('data/fbank'), 'max_duration': 30.0, 'bucketing_
↪sampler': False, 'num_buckets': 10, 'concatenate_cuts': False, 'duration_factor': 1.0,
↪'gap': 1.0, 'on_the_fly_feats': False, 'shuffle': False, 'return_cuts': True, 'num_
↪workers': 2, 'env_info': {'k2-version': '1.24.3', 'k2-build-type': 'Release', 'k2-with-
↪cuda': True, 'k2-git-sha1': '4c05309499a08454997adf500b56dcc629e35ae5', 'k2-git-date':
↪'Tue Jul 25 16:23:36 2023', 'lhotse-version': '1.16.0.dev+git.7640d66.clean', 'torch-
↪version': '1.13.0+cu116', 'torch-cuda-available': False, 'torch-cuda-version': '11.6',
↪'python-version': '3.8', 'icefall-git-branch': 'master', 'icefall-git-sha1': '3fb0a43-
```

(continues on next page)

(continued from previous page)

```
↪clean', 'icefall-git-date': 'Thu Jul 27 12:36:05 2023', 'icefall-path': '/tmp/icefall',
↪ 'k2-path': '/star-fj/fangjun/test-icefall/lib/python3.8/site-packages/k2/__init__.py',
↪ 'lhotse-path': '/star-fj/fangjun/test-icefall/lib/python3.8/site-packages/lhotse/__
↪init__.py', 'hostname': 'de-74279-k2-train-1-1220091118-57c4d55446-sph26', 'IP address
↪': '10.177.77.20'}}
2023-07-27 12:50:51,941 INFO [lexicon.py:168] Loading pre-compiled data/lang_phone/Linv.
↪pt
2023-07-27 12:50:51,949 INFO [train.py:495] device: cpu
2023-07-27 12:50:51,965 INFO [asr_datamodule.py:146] About to get train cuts
2023-07-27 12:50:51,965 INFO [asr_datamodule.py:244] About to get train cuts
2023-07-27 12:50:51,967 INFO [asr_datamodule.py:149] About to create train dataset
2023-07-27 12:50:51,967 INFO [asr_datamodule.py:199] Using SingleCutSampler.
2023-07-27 12:50:51,967 INFO [asr_datamodule.py:205] About to create train dataloader
2023-07-27 12:50:51,968 INFO [asr_datamodule.py:218] About to get test cuts
2023-07-27 12:50:51,968 INFO [asr_datamodule.py:252] About to get test cuts
2023-07-27 12:50:52,565 INFO [train.py:422] Epoch 0, batch 0, loss[loss=1.065, over 2436.
↪00 frames. ], tot_loss[loss=1.065, over 2436.00 frames. ], batch size: 4
2023-07-27 12:50:53,681 INFO [train.py:422] Epoch 0, batch 10, loss[loss=0.4561, over␣
↪2828.00 frames. ], tot_loss[loss=0.7076, over 22192.90 frames.], batch size: 4
2023-07-27 12:50:54,167 INFO [train.py:444] Epoch 0, validation loss=0.9002, over 18067.
↪00 frames.
2023-07-27 12:50:55,011 INFO [train.py:422] Epoch 0, batch 20, loss[loss=0.2555, over␣
↪2695.00 frames. ], tot_loss[loss=0.484, over 34971.47 frames. ], batch size: 5
2023-07-27 12:50:55,331 INFO [train.py:444] Epoch 0, validation loss=0.4688, over 18067.
↪00 frames.
2023-07-27 12:50:55,368 INFO [checkpoint.py:75] Saving checkpoint to tdnn/exp/epoch-0.pt
2023-07-27 12:50:55,633 INFO [train.py:422] Epoch 1, batch 0, loss[loss=0.2532, over␣
↪2436.00 frames. ], tot_loss[loss=0.2532, over 2436.00 frames. ],
 batch size: 4
2023-07-27 12:50:56,242 INFO [train.py:422] Epoch 1, batch 10, loss[loss=0.1139, over␣
↪2828.00 frames. ], tot_loss[loss=0.1592, over 22192.90 frames.], batch size: 4
2023-07-27 12:50:56,522 INFO [train.py:444] Epoch 1, validation loss=0.1627, over 18067.
↪00 frames.
2023-07-27 12:50:57,209 INFO [train.py:422] Epoch 1, batch 20, loss[loss=0.07055, over␣
↪2695.00 frames. ], tot_loss[loss=0.1175, over 34971.47 frames.], batch size: 5
2023-07-27 12:50:57,600 INFO [train.py:444] Epoch 1, validation loss=0.07091, over 18067.
↪00 frames.
2023-07-27 12:50:57,640 INFO [checkpoint.py:75] Saving checkpoint to tdnn/exp/epoch-1.pt
2023-07-27 12:50:57,847 INFO [train.py:422] Epoch 2, batch 0, loss[loss=0.07731, over␣
↪2436.00 frames. ], tot_loss[loss=0.07731, over 2436.00 frames.], batch size: 4
2023-07-27 12:50:58,427 INFO [train.py:422] Epoch 2, batch 10, loss[loss=0.04391, over␣
↪2828.00 frames. ], tot_loss[loss=0.05341, over 22192.90 frames. ], batch size: 4
2023-07-27 12:50:58,884 INFO [train.py:444] Epoch 2, validation loss=0.04384, over 18067.
↪00 frames.
2023-07-27 12:50:59,387 INFO [train.py:422] Epoch 2, batch 20, loss[loss=0.03458, over␣
↪2695.00 frames. ], tot_loss[loss=0.04616, over 34971.47 frames. ], batch size: 5
2023-07-27 12:50:59,707 INFO [train.py:444] Epoch 2, validation loss=0.03379, over 18067.
↪00 frames.
2023-07-27 12:50:59,758 INFO [checkpoint.py:75] Saving checkpoint to tdnn/exp/epoch-2.pt

  ... ...
```

(continues on next page)

(continued from previous page)

```
2023-07-27 12:51:23,433 INFO [train.py:422] Epoch 13, batch 0, loss[loss=0.01054, over␣
↪2436.00 frames. ], tot_loss[loss=0.01054, over 2436.00 frames. ], batch size: 4
2023-07-27 12:51:23,980 INFO [train.py:422] Epoch 13, batch 10, loss[loss=0.009014, over␣
↪2828.00 frames. ], tot_loss[loss=0.009974, over 22192.90 frames. ], batch size: 4
2023-07-27 12:51:24,489 INFO [train.py:444] Epoch 13, validation loss=0.01085, over␣
↪18067.00 frames.
2023-07-27 12:51:25,258 INFO [train.py:422] Epoch 13, batch 20, loss[loss=0.01172, over␣
↪2695.00 frames. ], tot_loss[loss=0.01055, over 34971.47 frames. ], batch size: 5
2023-07-27 12:51:25,621 INFO [train.py:444] Epoch 13, validation loss=0.01074, over␣
↪18067.00 frames.
2023-07-27 12:51:25,699 INFO [checkpoint.py:75] Saving checkpoint to tdnn/exp/epoch-13.pt
2023-07-27 12:51:25,866 INFO [train.py:422] Epoch 14, batch 0, loss[loss=0.01044, over␣
↪2436.00 frames. ], tot_loss[loss=0.01044, over 2436.00 frames. ], batch size: 4
2023-07-27 12:51:26,844 INFO [train.py:422] Epoch 14, batch 10, loss[loss=0.008942, over␣
↪2828.00 frames. ], tot_loss[loss=0.01, over 22192.90 frames. ], batch size: 4
2023-07-27 12:51:27,221 INFO [train.py:444] Epoch 14, validation loss=0.01082, over␣
↪18067.00 frames.
2023-07-27 12:51:27,970 INFO [train.py:422] Epoch 14, batch 20, loss[loss=0.01169, over␣
↪2695.00 frames. ], tot_loss[loss=0.01054, over 34971.47 frames. ], batch size: 5
2023-07-27 12:51:28,247 INFO [train.py:444] Epoch 14, validation loss=0.01073, over␣
↪18067.00 frames.
2023-07-27 12:51:28,323 INFO [checkpoint.py:75] Saving checkpoint to tdnn/exp/epoch-14.pt
2023-07-27 12:51:28,326 INFO [train.py:555] Done!
```

2.7.3 Decoding

Let us use the trained model to decode the test set:

```
(test-icefall) kuangfangjun:ASR$ ./tdnn/decode.py

2023-07-27 12:55:12,840 INFO [decode.py:263] Decoding started
2023-07-27 12:55:12,840 INFO [decode.py:264] {'exp_dir': PosixPath('tdnn/exp'), 'lang_dir
↪': PosixPath('data/lang_phone'), 'lm_dir': PosixPath('data/lm'), 'feature_dim': 23,
↪'search_beam': 20, 'output_beam': 8, 'min_active_states': 30, 'max_active_states':␣
↪10000, 'use_double_scores': True, 'epoch': 14, 'avg': 2, 'export': False, 'feature_dir
↪': PosixPath('data/fbank'), 'max_duration': 30.0, 'bucketing_sampler': False, 'num_
↪buckets': 10, 'concatenate_cuts': False, 'duration_factor': 1.0, 'gap': 1.0, 'on_the_
↪fly_feats': False, 'shuffle': False, 'return_cuts': True, 'num_workers': 2, 'env_info
↪': {'k2-version': '1.24.3', 'k2-build-type': 'Release', 'k2-with-cuda': True, 'k2-git-
↪sha1': '4c05309499a08454997adf500b56dcc629e35ae5', 'k2-git-date': 'Tue Jul 25 16:23:36␣
↪2023', 'lhotse-version': '1.16.0.dev+git.7640d66.clean', 'torch-version': '1.13.0+cu116
↪', 'torch-cuda-available': False, 'torch-cuda-version': '11.6', 'python-version': '3.8
↪', 'icefall-git-branch': 'master', 'icefall-git-sha1': '3fb0a43-clean', 'icefall-git-
↪date': 'Thu Jul 27 12:36:05 2023', 'icefall-path': '/tmp/icefall', 'k2-path': '/star-
↪fj/fangjun/test-icefall/lib/python3.8/site-packages/k2/__init__.py', 'lhotse-path': '/
↪star-fj/fangjun/test-icefall/lib/python3.8/site-packages/lhotse/__init__.py', 'hostname
↪': 'de-74279-k2-train-1-1220091118-57c4d55446-sph26', 'IP address': '10.177.77.20'}}
2023-07-27 12:55:12,841 INFO [lexicon.py:168] Loading pre-compiled data/lang_phone/Linv.
↪pt
2023-07-27 12:55:12,855 INFO [decode.py:273] device: cpu
2023-07-27 12:55:12,868 INFO [decode.py:291] averaging ['tdnn/exp/epoch-13.pt', 'tdnn/
```

(continues on next page)

(continued from previous page)

```
↪exp/epoch-14.pt']
2023-07-27 12:55:12,882 INFO [asr_datamodule.py:218] About to get test cuts
2023-07-27 12:55:12,883 INFO [asr_datamodule.py:252] About to get test cuts
2023-07-27 12:55:13,157 INFO [decode.py:204] batch 0/?, cuts processed until now is 4
2023-07-27 12:55:13,701 INFO [decode.py:241] The transcripts are stored in tdnn/exp/
↪recogs-test_set.txt
2023-07-27 12:55:13,702 INFO [utils.py:564] [test_set] %WER 0.42% [1 / 240, 0 ins, 1 del,
↪ 0 sub ]
2023-07-27 12:55:13,704 INFO [decode.py:249] Wrote detailed error stats to tdnn/exp/errs-
↪test_set.txt
2023-07-27 12:55:13,704 INFO [decode.py:316] Done!
```

Congratulations! You have successfully setup the environment and have run the first recipe in icefall.

Have fun with `icefall`!

2.8 YouTube Video

We provide the following YouTube video showing how to install icefall. It also shows how to debug various problems that you may encounter while using icefall.

Note: To get the latest news of next-gen Kaldi, please subscribe the following YouTube channel by Nadira Povey:

https://www.youtube.com/channel/UC_VaumpkmINz1pNkFXAN9mw

https://youtu.be/LVmrBD0tLfE

CHAPTER

THREE

DOCKER

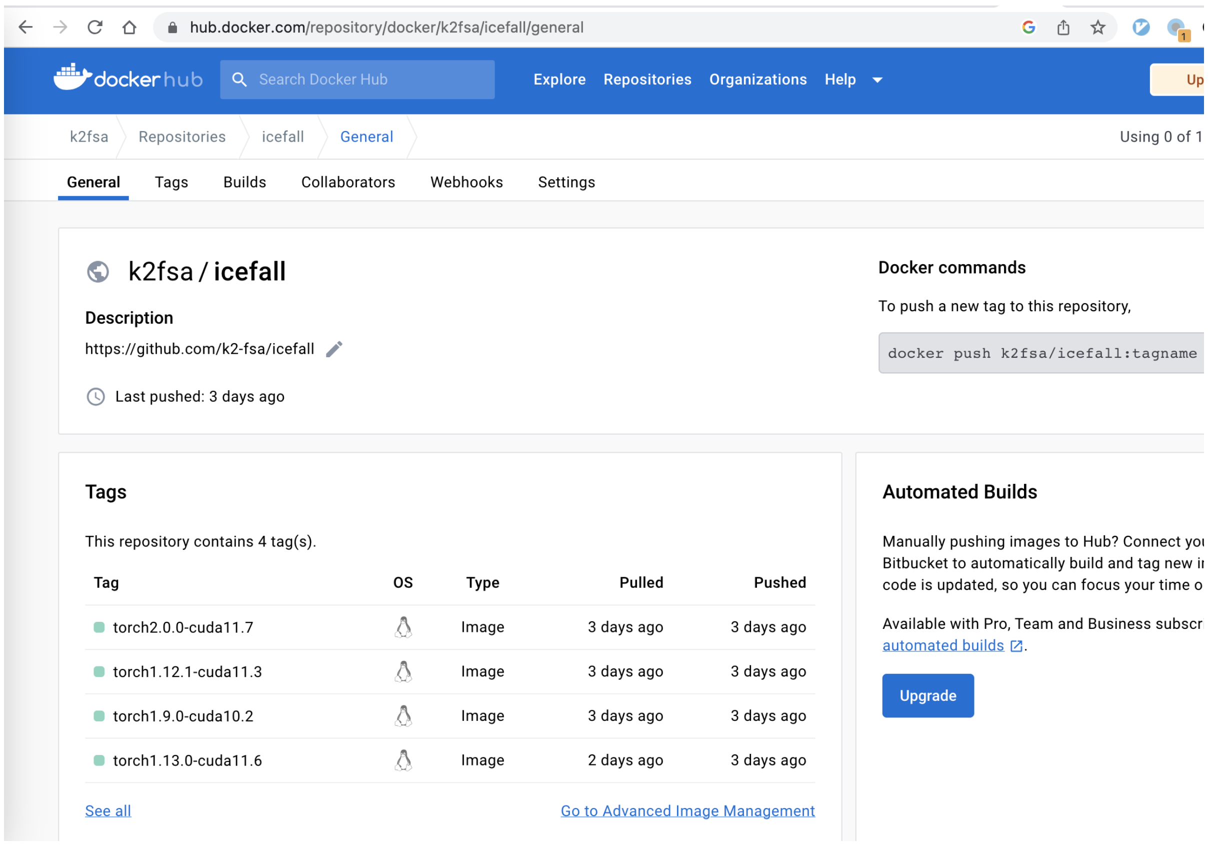
This section describes how to use pre-built docker images to run icefall.

Hint: If you only have CPUs available, you can still use the pre-built docker images.

3.1 Introduction

We have pre-built docker images hosted at the following address:

https://hub.docker.com/repository/docker/k2fsa/icefall/general



k2fsa / icefall

Description

https://github.com/k2-fsa/icefall

Last pushed: 3 days ago

Docker commands

To push a new tag to this repository,

```
docker push k2fsa/icefall:tagname
```

Tags

This repository contains 4 tag(s).

Tag	OS	Type	Pulled	Pushed
torch2.0.0-cuda11.7		Image	3 days ago	3 days ago
torch1.12.1-cuda11.3		Image	3 days ago	3 days ago
torch1.9.0-cuda10.2		Image	3 days ago	3 days ago
torch1.13.0-cuda11.6		Image	2 days ago	3 days ago

Automated Builds

Manually pushing images to Hub? Connect yo
Bitbucket to automatically build and tag new i
code is updated, so you can focus your time o

Available with Pro, Team and Business subscr
automated builds.

You can find the `Dockerfile` at https://github.com/k2-fsa/icefall/tree/master/docker.

We describe the following items in this section:

- How to view available tags
- How to download pre-built docker images
- How to run the yesno recipe within a docker container on `CPU`

3.2 View available tags

You can use the following command to view available tags:

```
curl -s 'https://registry.hub.docker.com/v2/repositories/k2fsa/icefall/tags/'|jq '.
↪"results"[]["name"]'
```

which will give you something like below:

```
"torch2.0.0-cuda11.7"
"torch1.12.1-cuda11.3"
"torch1.9.0-cuda10.2"
"torch1.13.0-cuda11.6"
```

Hint: Available tags will be updated when there are new releases of torch.

Please select an appropriate combination of torch and CUDA.

3.3 Download a docker image

Suppose that you select the tag `torch1.13.0-cuda11.6`, you can use the following command to download it:

```
sudo docker image pull k2fsa/icefall:torch1.13.0-cuda11.6
```

3.4 Run a docker image with GPU

```
sudo docker run --gpus all --rm -it k2fsa/icefall:torch1.13.0-cuda11.6 /bin/bash
```

3.5 Run a docker image with CPU

```
sudo docker run --rm -it k2fsa/icefall:torch1.13.0-cuda11.6 /bin/bash
```

3.6 Run yesno within a docker container

After starting the container, the following interface is presented:

```
root@60c947eac59c:/workspace/icefall#
```

It shows the current user is `root` and the current working directory is `/workspace/icefall`.

3.6.1 Update the code

Please first run:

```
root@60c947eac59c:/workspace/icefall# git pull
```

so that your local copy contains the latest code.

3.6.2 Data preparation

Now we can use

```
root@60c947eac59c:/workspace/icefall# cd egs/yesno/ASR/
```

to switch to the `yesno` recipe and run

```
root@60c947eac59c:/workspace/icefall/egs/yesno/ASR# ./prepare.sh
```

Hint:

If you are running without GPU, it may report the following error:

```
File "/opt/conda/lib/python3.9/site-packages/k2/__init__.py", line 23,␣
↪in <module>
  from _k2 import DeterminizeWeightPushingType
ImportError: libcuda.so.1: cannot open shared object file: No such␣
↪file or directory
```

We can use the following command to fix it:

```
root@60c947eac59c:/workspace/icefall/egs/yesno/ASR# ln -s /opt/conda/lib/stubs/
↪libcuda.so /opt/conda/lib/stubs/libcuda.so.1
```

The logs of running `./prepare.sh` are listed below:

3.6.3 Training

After preparing the data, we can start training with the following command

```
root@60c947eac59c:/workspace/icefall/egs/yesno/ASR# ./tdnn/train.py
```

All of the training logs are given below:

Hint: It is running on CPU and it takes only 16 seconds for this run.

3.6.4 Decoding

After training, we can decode the trained model with

```
root@60c947eac59c:/workspace/icefall/egs/yesno/ASR# ./tdnn/decode.py
```

The decoding logs are given below:

```
2023-08-01 02:06:22,400 INFO [decode.py:263] Decoding started
2023-08-01 02:06:22,400 INFO [decode.py:264] {'exp_dir': PosixPath('tdnn/exp'), 'lang_dir
↪': PosixPath('data/lang_phone'), 'lm_dir': PosixPath('data/lm'), 'feature_dim': 23,
↪'search_beam': 20, 'output_beam': 8, 'min_active_states': 30, 'max_active_states':␣
↪10000, 'use_double_scores': True, 'epoch': 14, 'avg': 2, 'export': False, 'feature_dir
↪': PosixPath('data/fbank'), 'max_duration': 30.0, 'bucketing_sampler': False, 'num_
↪buckets': 10, 'concatenate_cuts': False, 'duration_factor': 1.0, 'gap': 1.0, 'on_the_
↪fly_feats': False, 'shuffle': False, 'return_cuts': True, 'num_workers': 2, 'env_info
↪': {'k2-version': '1.24.3', 'k2-build-type': 'Release', 'k2-with-cuda': True, 'k2-git-
↪sha1': '4c05309499a08454997adf500b56dcc629e35ae5', 'k2-git-date': 'Tue Jul 25 16:23:36␣
↪2023', 'lhotse-version': '1.16.0.dev+git.7640d663.clean', 'torch-version': '1.13.0',
↪'torch-cuda-available': False, 'torch-cuda-version': '11.6', 'python-version': '3.9',
↪'icefall-git-branch': 'master', 'icefall-git-sha1': '375520d-clean', 'icefall-git-date
↪': 'Fri Jul 28 07:43:08 2023', 'icefall-path': '/workspace/icefall', 'k2-path': '/opt/
↪conda/lib/python3.9/site-packages/k2/__init__.py', 'lhotse-path': '/opt/conda/lib/
↪python3.9/site-packages/lhotse/__init__.py', 'hostname': '60c947eac59c', 'IP address':
↪'172.17.0.2'}}
2023-08-01 02:06:22,401 INFO [lexicon.py:168] Loading pre-compiled data/lang_phone/Linv.
↪pt
2023-08-01 02:06:22,403 INFO [decode.py:273] device: cpu
2023-08-01 02:06:22,406 INFO [decode.py:291] averaging ['tdnn/exp/epoch-13.pt', 'tdnn/
↪exp/epoch-14.pt']
2023-08-01 02:06:22,424 INFO [asr_datamodule.py:218] About to get test cuts
2023-08-01 02:06:22,425 INFO [asr_datamodule.py:252] About to get test cuts
2023-08-01 02:06:22,504 INFO [decode.py:204] batch 0/?, cuts processed until now is 4
[W NNPACK.cpp:53] Could not initialize NNPACK! Reason: Unsupported hardware.
2023-08-01 02:06:22,687 INFO [decode.py:241] The transcripts are stored in tdnn/exp/
↪recogs-test_set.txt
2023-08-01 02:06:22,688 INFO [utils.py:564] [test_set] %WER 0.42% [1 / 240, 0 ins, 1 del,
↪ 0 sub ]
2023-08-01 02:06:22,690 INFO [decode.py:249] Wrote detailed error stats to tdnn/exp/errs-
↪test_set.txt
2023-08-01 02:06:22,690 INFO [decode.py:316] Done!
```

Congratulations! You have finished successfully running icefall within a docker container.

CHAPTER

FOUR

FREQUENTLY ASKED QUESTIONS (FAQS)

In this section, we collect issues reported by users and post the corresponding solutions.

4.1 OSError: libtorch_hip.so: cannot open shared object file: no such file or directory

One user is using the following code to install `torch` and `torchaudio`:

```
pip install \
  torch==1.10.0+cu111 \
  torchvision==0.11.0+cu111 \
  torchaudio==0.10.0 \
  -f https://download.pytorch.org/whl/torch_stable.html
```

and it throws the following error when running `tdnn/train.py`:

```
OSError: libtorch_hip.so: cannot open shared object file: no such file or directory
```

The fix is to specify the CUDA version while installing `torchaudio`. That is, change `torchaudio==0.10.0` to `torchaudio==0.10.0+cu11``. Therefore, the correct command is:

```
pip install \
  torch==1.10.0+cu111 \
  torchvision==0.11.0+cu111 \
  torchaudio==0.10.0+cu111 \
  -f https://download.pytorch.org/whl/torch_stable.html
```

4.2 AttributeError: module 'distutils' has no attribute 'version'

The error log is:

```
Traceback (most recent call last):
  File "./tdnn/train.py", line 14, in <module>
    from asr_datamodule import YesNoAsrDataModule
  File "/home/xxx/code/next-gen-kaldi/icefall/egs/yesno/ASR/tdnn/asr_datamodule.py",␣
↪line 34, in <module>
    from icefall.dataset.datamodule import DataModule
  File "/home/xxx/code/next-gen-kaldi/icefall/icefall/__init__.py", line 3, in <module>
```

(continues on next page)

(continued from previous page)

```
    from . import (
  File "/home/xxx/code/next-gen-kaldi/icefall/icefall/decode.py", line 23, in <module>
    from icefall.utils import add_eos, add_sos, get_texts
  File "/home/xxx/code/next-gen-kaldi/icefall/icefall/utils.py", line 39, in <module>
    from torch.utils.tensorboard import SummaryWriter
  File "/home/xxx/tool/miniconda3/envs/yyy/lib/python3.8/site-packages/torch/utils/
↪tensorboard/__init__.py", line 4, in <module>
    LooseVersion = distutils.version.LooseVersion
AttributeError: module 'distutils' has no attribute 'version'
```

The fix is:

```
pip uninstall setuptools

pip install setuptools==58.0.4
```

4.3 ImportError: libpython3.10.so.1.0: cannot open shared object file: No such file or directory

If you are using conda and encounter the following issue:

```
Traceback (most recent call last):
  File "/k2-dev/yangyifan/anaconda3/envs/icefall/lib/python3.10/site-packages/k2-1.23.3.
↪dev20230112+cuda11.6.torch1.13.1-py3.10-linux-x86_64.egg/k2/__init__.py", line 24, in
↪<module>
    from _k2 import DeterminizeWeightPushingType
ImportError: libpython3.10.so.1.0: cannot open shared object file: No such file or␣
↪directory

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "/k2-dev/yangyifan/icefall/egs/librispeech/ASR/./pruned_transducer_stateless7_ctc_
↪bs/decode.py", line 104, in <module>
    import k2
  File "/k2-dev/yangyifan/anaconda3/envs/icefall/lib/python3.10/site-packages/k2-1.23.3.
↪dev20230112+cuda11.6.torch1.13.1-py3.10-linux-x86_64.egg/k2/__init__.py", line 30, in
↪<module>
    raise ImportError(
ImportError: libpython3.10.so.1.0: cannot open shared object file: No such file or␣
↪directory
Note: If you're using anaconda and importing k2 on MacOS,
      you can probably fix this by setting the environment variable:
  export DYLD_LIBRARY_PATH=$CONDA_PREFIX/lib/python3.10/site-packages:$DYLD_LIBRARY_PATH
```

Please first try to find where `libpython3.10.so.1.0` locates.

For instance,

```
cd $CONDA_PREFIX/lib
find . -name "libpython*"
```

If you are able to find it inside `$CODNA_PREFIX/lib`, please set the following environment variable:

```
export LD_LIBRARY_PATH=$CONDA_PREFIX/lib:$LD_LIBRARY_PATH
```

CHAPTER

FIVE

MODEL EXPORT

In this section, we describe various ways to export models.

5.1 Export model.state_dict()

5.1.1 When to use it

During model training, we save checkpoints periodically to disk.

A checkpoint contains the following information:

- `model.state_dict()`
- `optimizer.state_dict()`
- and some other information related to training

When we need to resume the training process from some point, we need a checkpoint. However, if we want to publish the model for inference, then only `model.state_dict()` is needed. In this case, we need to strip all other information except `model.state_dict()` to reduce the file size of the published model.

5.1.2 How to export

Every recipe contains a file `export.py` that you can use to export `model.state_dict()` by taking some checkpoints as inputs.

Hint: Each `export.py` contains well-documented usage information.

In the following, we use https://github.com/k2-fsa/icefall/blob/master/egs/librispeech/ASR/pruned_transducer_stateless3/export.py as an example.

Note: The steps for other recipes are almost the same.

```
cd egs/librispeech/ASR

./pruned_transducer_stateless3/export.py \
  --exp-dir ./pruned_transducer_stateless3/exp \
  --tokens data/lang_bpe_500/tokens.txt \
```

(continues on next page)

(continued from previous page)

```
  --epoch 20 \
  --avg 10
```

will generate a file `pruned_transducer_stateless3/exp/pretrained.pt`, which is a dict containing `{"model": model.state_dict()}` saved by `torch.save()`.

5.1.3 How to use the exported model

For each recipe, we provide pretrained models hosted on huggingface. You can find links to pretrained models in `RESULTS.md` of each dataset.

In the following, we demonstrate how to use the pretrained model from https://huggingface.co/csukuangfj/icefall-asr-librispeech-pruned-transducer-stateless3-2022-05-13.

```
cd egs/librispeech/ASR

git lfs install
git clone https://huggingface.co/csukuangfj/icefall-asr-librispeech-pruned-transducer-
↪stateless3-2022-05-13
```

After cloning the repo with `git lfs`, you will find several files in the folder `icefall-asr-librispeech-pruned-transducer-stateless3-2022-05-13/exp` that have a prefix `pretrained-`. Those files contain `model.state_dict()` exported by the above `export.py`.

In each recipe, there is also a file `pretrained.py`, which can use `pretrained-xxx.pt` to decode waves. The following is an example:

```
cd egs/librispeech/ASR

./pruned_transducer_stateless3/pretrained.py \
   --checkpoint ./icefall-asr-librispeech-pruned-transducer-stateless3-2022-05-13/exp/
↪pretrained-iter-1224000-avg-14.pt \
   --tokens ./icefall-asr-librispeech-pruned-transducer-stateless3-2022-05-13/data/lang_
↪bpe_500/tokens.txt \
   --method greedy_search \
   ./icefall-asr-librispeech-pruned-transducer-stateless3-2022-05-13/test_wavs/1089-
↪134686-0001.wav \
   ./icefall-asr-librispeech-pruned-transducer-stateless3-2022-05-13/test_wavs/1221-
↪135766-0001.wav \
   ./icefall-asr-librispeech-pruned-transducer-stateless3-2022-05-13/test_wavs/1221-
↪135766-0002.wav
```

The above commands show how to use the exported model with `pretrained.py` to decode multiple sound files. Its output is given as follows for reference:

```
2022-10-13 19:09:02,233 INFO [pretrained.py:265] {'best_train_loss': inf, 'best_valid_
↪loss': inf, 'best_train_epoch': -1, 'best_valid_epoch': -1, 'batch_idx_train': 0, 'log_
↪interval': 50, 'reset_interval': 200, 'valid_interval': 3000, 'feature_dim': 80,
↪'subsampling_factor': 4, 'encoder_dim': 512, 'nhead': 8, 'dim_feedforward': 2048, 'num_
↪encoder_layers': 12, 'decoder_dim': 512, 'joiner_dim': 512, 'model_warm_step': 3000,
↪'env_info': {'k2-version': '1.21', 'k2-build-type': 'Release', 'k2-with-cuda': True,
↪'k2-git-sha1': '4810e00d8738f1a21278b0156a42ff396a2d40ac', 'k2-git-date': 'Fri Oct 7␣
↪19:35:03 2022', 'lhotse-version': '1.3.0.dev+missing.version.file', 'torch-version':
```

(continues on next page)

(continued from previous page)

```
↪'1.10.0+cu102', 'torch-cuda-available': False, 'torch-cuda-version': '10.2', 'python-
↪version': '3.8', 'icefall-git-branch': 'onnx-doc-1013', 'icefall-git-sha1': 'c39cba5-
↪dirty', 'icefall-git-date': 'Thu Oct 13 15:17:20 2022', 'icefall-path': '/k2-dev/
↪fangjun/open-source/icefall-master', 'k2-path': '/k2-dev/fangjun/open-source/k2-master/
↪k2/python/k2/__init__.py', 'lhotse-path': '/ceph-fj/fangjun/open-source-2/lhotse-jsonl/
↪lhotse/__init__.py', 'hostname': 'de-74279-k2-test-4-0324160024-65bfd8b584-jjlbn', 'IP␣
↪address': '10.177.74.203'}, 'checkpoint': './icefall-asr-librispeech-pruned-transducer-
↪stateless3-2022-05-13/exp/pretrained-iter-1224000-avg-14.pt', 'bpe_model': './icefall-
↪asr-librispeech-pruned-transducer-stateless3-2022-05-13/data/lang_bpe_500/bpe.model',
↪'method': 'greedy_search', 'sound_files': ['./icefall-asr-librispeech-pruned-
↪transducer-stateless3-2022-05-13/test_wavs/1089-134686-0001.wav', './icefall-asr-
↪librispeech-pruned-transducer-stateless3-2022-05-13/test_wavs/1221-135766-0001.wav', '.
↪/icefall-asr-librispeech-pruned-transducer-stateless3-2022-05-13/test_wavs/1221-135766-
↪0002.wav'], 'sample_rate': 16000, 'beam_size': 4, 'beam': 4, 'max_contexts': 4, 'max_
↪states': 8, 'context_size': 2, 'max_sym_per_frame': 1, 'simulate_streaming': False,
↪'decode_chunk_size': 16, 'left_context': 64, 'dynamic_chunk_training': False, 'causal_
↪convolution': False, 'short_chunk_size': 25, 'num_left_chunks': 4, 'blank_id': 0, 'unk_
↪id': 2, 'vocab_size': 500}
2022-10-13 19:09:02,233 INFO [pretrained.py:271] device: cpu
2022-10-13 19:09:02,233 INFO [pretrained.py:273] Creating model
2022-10-13 19:09:02,612 INFO [train.py:458] Disable giga
2022-10-13 19:09:02,623 INFO [pretrained.py:277] Number of model parameters: 78648040
2022-10-13 19:09:02,951 INFO [pretrained.py:285] Constructing Fbank computer
2022-10-13 19:09:02,952 INFO [pretrained.py:295] Reading sound files: ['./icefall-asr-
↪librispeech-pruned-transducer-stateless3-2022-05-13/test_wavs/1089-134686-0001.wav', '.
↪/icefall-asr-librispeech-pruned-transducer-stateless3-2022-05-13/test_wavs/1221-135766-
↪0001.wav', './icefall-asr-librispeech-pruned-transducer-stateless3-2022-05-13/test_
↪wavs/1221-135766-0002.wav']
2022-10-13 19:09:02,957 INFO [pretrained.py:301] Decoding started
2022-10-13 19:09:06,700 INFO [pretrained.py:329] Using greedy_search
2022-10-13 19:09:06,912 INFO [pretrained.py:388]
./icefall-asr-librispeech-pruned-transducer-stateless3-2022-05-13/test_wavs/1089-134686-
↪0001.wav:
AFTER EARLY NIGHTFALL THE YELLOW LAMPS WOULD LIGHT UP HERE AND THERE THE SQUALID QUARTER␣
↪OF THE BROTHELS

./icefall-asr-librispeech-pruned-transducer-stateless3-2022-05-13/test_wavs/1221-135766-
↪0001.wav:
GOD AS A DIRECT CONSEQUENCE OF THE SIN WHICH MAN THUS PUNISHED HAD GIVEN HER A LOVELY␣
↪CHILD WHOSE PLACE WAS ON THAT SAME DISHONORED BOSOM TO CONNECT HER PARENT FOREVER WITH␣
↪THE RACE AND DESCENT OF MORTALS AND TO BE FINALLY A BLESSED SOUL IN HEAVEN

./icefall-asr-librispeech-pruned-transducer-stateless3-2022-05-13/test_wavs/1221-135766-
↪0002.wav:
YET THESE THOUGHTS AFFECTED HESTER PRYNNE LESS WITH HOPE THAN APPREHENSION


2022-10-13 19:09:06,912 INFO [pretrained.py:390] Decoding Done
```

5.1.4 Use the exported model to run decode.py

When we publish the model, we always note down its WERs on some test dataset in `RESULTS.md`. This section describes how to use the pretrained model to reproduce the WER.

```
cd egs/librispeech/ASR
git lfs install
git clone https://huggingface.co/csukuangfj/icefall-asr-librispeech-pruned-transducer-
↪stateless3-2022-05-13

cd icefall-asr-librispeech-pruned-transducer-stateless3-2022-05-13/exp
ln -s pretrained-iter-1224000-avg-14.pt epoch-9999.pt
cd ../..
```

We create a symlink with name `epoch-9999.pt` to `pretrained-iter-1224000-avg-14.pt`, so that we can pass `--epoch 9999 --avg 1` to `decode.py` in the following command:

```
./pruned_transducer_stateless3/decode.py \
    --epoch 9999 \
    --avg 1 \
    --exp-dir ./icefall-asr-librispeech-pruned-transducer-stateless3-2022-05-13/exp \
    --lang-dir ./icefall-asr-librispeech-pruned-transducer-stateless3-2022-05-13/data/
↪lang_bpe_500 \
    --max-duration 600 \
    --decoding-method greedy_search
```

You will find the decoding results in `./icefall-asr-librispeech-pruned-transducer-stateless3-2022-05-13/exp/greedy_search`.

Caution: For some recipes, you also need to pass `--use-averaged-model False` to `decode.py`. The reason is that the exported pretrained model is already the averaged one.

Hint: Before running `decode.py`, we assume that you have already run `prepare.sh` to prepare the test dataset.

5.2 Export model with torch.jit.trace()

In this section, we describe how to export a model via `torch.jit.trace()`.

5.2.1 When to use it

If we want to use our trained model with torchscript, we can use `torch.jit.trace()`.

Hint: See *Export model with torch.jit.script()* if you want to use `torch.jit.script()`.

5.2.2 How to export

We use https://github.com/k2-fsa/icefall/tree/master/egs/librispeech/ASR/lstm_transducer_stateless2 as an example in the following.

```
iter=468000
avg=16

cd egs/librispeech/ASR

./lstm_transducer_stateless2/export.py \
  --exp-dir ./lstm_transducer_stateless2/exp \
  --tokens data/lang_bpe_500/tokens.txt \
  --iter $iter \
  --avg  $avg \
  --jit-trace 1
```

It will generate three files inside `lstm_transducer_stateless2/exp`:

- `encoder_jit_trace.pt`
- `decoder_jit_trace.pt`
- `joiner_jit_trace.pt`

You can use https://github.com/k2-fsa/icefall/blob/master/egs/librispeech/ASR/lstm_transducer_stateless2/jit_pretrained.py to decode sound files with the following commands:

```
cd egs/librispeech/ASR
./lstm_transducer_stateless2/jit_pretrained.py \
  --bpe-model ./data/lang_bpe_500/bpe.model \
  --encoder-model-filename ./lstm_transducer_stateless2/exp/encoder_jit_trace.pt \
  --decoder-model-filename ./lstm_transducer_stateless2/exp/decoder_jit_trace.pt \
  --joiner-model-filename ./lstm_transducer_stateless2/exp/joiner_jit_trace.pt \
  /path/to/foo.wav \
  /path/to/bar.wav \
  /path/to/baz.wav
```

5.2.3 How to use the exported models

Please refer to https://k2-fsa.github.io/sherpa/python/streaming_asr/lstm/index.html for its usage in sherpa. You can also find pretrained models there.

5.3 Export model with torch.jit.script()

In this section, we describe how to export a model via `torch.jit.script()`.

5.3.1 When to use it

If we want to use our trained model with torchscript, we can use `torch.jit.script()`.

Hint: See *Export model with torch.jit.trace()* if you want to use `torch.jit.trace()`.

5.3.2 How to export

We use https://github.com/k2-fsa/icefall/tree/master/egs/librispeech/ASR/pruned_transducer_stateless3 as an example in the following.

```
cd egs/librispeech/ASR
epoch=14
avg=1

./pruned_transducer_stateless3/export.py \
  --exp-dir ./pruned_transducer_stateless3/exp \
  --tokens data/lang_bpe_500/tokens.txt \
  --epoch $epoch \
  --avg $avg \
  --jit 1
```

It will generate a file `cpu_jit.pt` in `pruned_transducer_stateless3/exp`.

Caution: Don't be confused by `cpu` in `cpu_jit.pt`. We move all parameters to CPU before saving it into a `pt` file; that's why we use `cpu` in the filename.

5.3.3 How to use the exported model

Please refer to the following pages for usage:

- https://k2-fsa.github.io/sherpa/python/streaming_asr/emformer/index.html
- https://k2-fsa.github.io/sherpa/python/streaming_asr/conv_emformer/index.html
- https://k2-fsa.github.io/sherpa/python/streaming_asr/conformer/index.html
- https://k2-fsa.github.io/sherpa/python/offline_asr/conformer/index.html
- https://k2-fsa.github.io/sherpa/cpp/offline_asr/gigaspeech.html
- https://k2-fsa.github.io/sherpa/cpp/offline_asr/wenetspeech.html

5.4 Export to ONNX

In this section, we describe how to export models to ONNX.

Hint: Before you continue, please run:

```
pip install onnx
```

In each recipe, there is a file called `export-onnx.py`, which is used to export trained models to ONNX.

There is also a file named `onnx_pretrained.py`, which you can use the exported ONNX model in Python with onnxruntime to decode sound files.

5.4.1 sherpa-onnx

We have a separate repository sherpa-onnx for deploying your exported models on various platforms such as:

- iOS
- Android
- Raspberry Pi
- Linux/macOS/Windows

Please see the documentation of sherpa-onnx for details:

https://k2-fsa.github.io/sherpa/onnx/index.html

5.4.2 Example

In the following, we demonstrate how to export a streaming Zipformer pre-trained model from https://huggingface.co/csukuangfj/icefall-asr-librispeech-pruned-transducer-stateless7-2022-11-11 to ONNX.

5.4.3 Download the pre-trained model

Hint: We assume you have installed git-lfs.

```
cd egs/librispeech/ASR

repo_url=https://huggingface.co/Zengwei/icefall-asr-librispeech-pruned-transducer-
↪stateless7-streaming-2022-12-29
GIT_LFS_SKIP_SMUDGE=1 git clone $repo_url
repo=$(basename $repo_url)

pushd $repo
git lfs pull --include "data/lang_bpe_500/bpe.model"
git lfs pull --include "exp/pretrained.pt"
cd exp
ln -s pretrained.pt epoch-99.pt
popd
```

5.4.4 Export the model to ONNX

```
./pruned_transducer_stateless7_streaming/export-onnx.py \
  --tokens $repo/data/lang_bpe_500/tokens.txt \
  --use-averaged-model 0 \
  --epoch 99 \
  --avg 1 \
  --decode-chunk-len 32 \
  --exp-dir $repo/exp/
```

Warning: `export-onnx.py` from different recipes has different options.

In the above example, `--decode-chunk-len` is specific for the streaming Zipformer. Other models won't have such an option.

It will generate the following 3 files in `$repo/exp`

- `encoder-epoch-99-avg-1.onnx`
- `decoder-epoch-99-avg-1.onnx`
- `joiner-epoch-99-avg-1.onnx`

5.4.5 Decode sound files with exported ONNX models

```
./pruned_transducer_stateless7_streaming/onnx_pretrained.py \
  --encoder-model-filename $repo/exp/encoder-epoch-99-avg-1.onnx \
  --decoder-model-filename $repo/exp/decoder-epoch-99-avg-1.onnx \
  --joiner-model-filename $repo/exp/joiner-epoch-99-avg-1.onnx \
  --tokens $repo/data/lang_bpe_500/tokens.txt \
  $repo/test_wavs/1089-134686-0001.wav
```

5.5 Export to ncnn

We support exporting the following models to ncnn:

- Zipformer transducer models
- LSTM transducer models
- ConvEmformer transducer models

We also provide sherpa-ncnn for performing speech recognition using ncnn with exported models. It has been tested on the following platforms:

- Linux
- macOS
- Windows
- `Android`
- `iOS`

- Raspberry Pi
- (MAIX-III AXera-Pi).
- RV1126

sherpa-ncnn is self-contained and can be statically linked to produce a binary containing everything needed. Please refer to its documentation for details:

- https://k2-fsa.github.io/sherpa/ncnn/index.html

5.5.1 Export streaming Zipformer transducer models to ncnn

We use the pre-trained model from the following repository as an example:

https://huggingface.co/Zengwei/icefall-asr-librispeech-pruned-transducer-stateless7-streaming-2022-12-29

We will show you step by step how to export it to ncnn and run it with sherpa-ncnn.

Hint: We use `Ubuntu 18.04`, `torch 1.13`, and `Python 3.8` for testing.

Caution: Please use a more recent version of PyTorch. For instance, `torch 1.8` may `not` work.

1. Download the pre-trained model

Hint: You have to install git-lfs before you continue.

```
cd egs/librispeech/ASR
GIT_LFS_SKIP_SMUDGE=1 git clone https://huggingface.co/Zengwei/icefall-asr-librispeech-
↪pruned-transducer-stateless7-streaming-2022-12-29
cd icefall-asr-librispeech-pruned-transducer-stateless7-streaming-2022-12-29

git lfs pull --include "exp/pretrained.pt"
git lfs pull --include "data/lang_bpe_500/bpe.model"

cd ..
```

Note: We downloaded `exp/pretrained-xxx.pt`, not `exp/cpu-jit_xxx.pt`.

In the above code, we downloaded the pre-trained model into the directory `egs/librispeech/ASR/icefall-asr-librispeech-pruned-transducer-stateless7-streaming-2022-12-29`.

2. Install ncnn and pnnx

Please refer to *2. Install ncnn and pnnx* .

3. Export the model via torch.jit.trace()

First, let us rename our pre-trained model:

```
cd egs/librispeech/ASR

cd icefall-asr-librispeech-pruned-transducer-stateless7-streaming-2022-12-29/exp

ln -s pretrained.pt epoch-99.pt

cd ../..
```

Next, we use the following code to export our model:

```
dir=./icefall-asr-librispeech-pruned-transducer-stateless7-streaming-2022-12-29

./pruned_transducer_stateless7_streaming/export-for-ncnn.py \
  --tokens $dir/data/lang_bpe_500/tokens.txt \
  --exp-dir $dir/exp \
  --use-averaged-model 0 \
  --epoch 99 \
  --avg 1 \
  --decode-chunk-len 32 \
  --num-left-chunks 4 \
  --num-encoder-layers "2,4,3,2,4" \
  --feedforward-dims "1024,1024,2048,2048,1024" \
  --nhead "8,8,8,8,8" \
  --encoder-dims "384,384,384,384,384" \
  --attention-dims "192,192,192,192,192" \
  --encoder-unmasked-dims "256,256,256,256,256" \
  --zipformer-downsampling-factors "1,2,4,8,2" \
  --cnn-module-kernels "31,31,31,31,31" \
  --decoder-dim 512 \
  --joiner-dim 512
```

Caution: If your model has different configuration parameters, please change them accordingly.

Hint: We have renamed our model to `epoch-99.pt` so that we can use `--epoch 99`. There is only one pre-trained model, so we use `--avg 1 --use-averaged-model 0`.

If you have trained a model by yourself and if you have all checkpoints available, please first use `decode.py` to tune `--epoch --avg` and select the best combination with with `--use-averaged-model 1`.

Note: You will see the following log output:

```
2023-02-27 20:23:07,473 INFO [export-for-ncnn.py:246] device: cpu
2023-02-27 20:23:07,477 INFO [export-for-ncnn.py:255] {'best_train_loss': inf, 'best_
↪valid_loss': inf, 'best_train_epoch': -1, 'best_valid_epoch': -1, 'batch_idx_train': 0,
↪ 'log_interval': 50, 'reset_interval': 200, 'valid_interval': 3000, 'feature_dim': 80,
↪'subsampling_factor': 4, 'warm_step': 2000, 'env_info': {'k2-version': '1.23.4', 'k2-
↪build-type': 'Release', 'k2-with-cuda': True, 'k2-git-sha1':
↪'62e404dd3f3a811d73e424199b3408e309c06e1a', 'k2-git-date': 'Mon Jan 30 10:26:16 2023',
↪'lhotse-version': '1.12.0.dev+missing.version.file', 'torch-version': '1.10.0+cu102',
↪'torch-cuda-available': True, 'torch-cuda-version': '10.2', 'python-version': '3.8',
↪'icefall-git-branch': 'master', 'icefall-git-sha1': '6d7a559-clean', 'icefall-git-date
↪': 'Thu Feb 16 19:47:54 2023', 'icefall-path': '/star-fj/fangjun/open-source/icefall-2
↪', 'k2-path': '/star-fj/fangjun/open-source/k2/k2/python/k2/__init__.py', 'lhotse-path
↪': '/star-fj/fangjun/open-source/lhotse/lhotse/__init__.py', 'hostname': 'de-74279-k2-
↪train-3-1220120619-7695ff496b-s9n4w', 'IP address': '10.177.6.147'}, 'epoch': 99, 'iter
↪': 0, 'avg': 1, 'exp_dir': PosixPath('icefall-asr-librispeech-pruned-transducer-
↪stateless7-streaming-2022-12-29/exp'), 'bpe_model': './icefall-asr-librispeech-pruned-
↪transducer-stateless7-streaming-2022-12-29/data/lang_bpe_500/bpe.model', 'context_size
↪': 2, 'use_averaged_model': False, 'num_encoder_layers': '2,4,3,2,4', 'feedforward_dims
↪': '1024,1024,2048,2048,1024', 'nhead': '8,8,8,8,8', 'encoder_dims': '384,384,384,384,
↪384', 'attention_dims': '192,192,192,192,192', 'encoder_unmasked_dims': '256,256,256,
↪256,256', 'zipformer_downsampling_factors': '1,2,4,8,2', 'cnn_module_kernels': '31,31,
↪31,31,31', 'decoder_dim': 512, 'joiner_dim': 512, 'short_chunk_size': 50, 'num_left_
↪chunks': 4, 'decode_chunk_len': 32, 'blank_id': 0, 'vocab_size': 500}
2023-02-27 20:23:07,477 INFO [export-for-ncnn.py:257] About to create model
2023-02-27 20:23:08,023 INFO [zipformer2.py:419] At encoder stack 4, which has␣
↪downsampling_factor=2, we will combine the outputs of layers 1 and 3, with␣
↪downsampling_factors=2 and 8.
2023-02-27 20:23:08,037 INFO [checkpoint.py:112] Loading checkpoint from icefall-asr-
↪librispeech-pruned-transducer-stateless7-streaming-2022-12-29/exp/epoch-99.pt
2023-02-27 20:23:08,655 INFO [export-for-ncnn.py:346] encoder parameters: 68944004
2023-02-27 20:23:08,655 INFO [export-for-ncnn.py:347] decoder parameters: 260096
2023-02-27 20:23:08,655 INFO [export-for-ncnn.py:348] joiner parameters: 716276
2023-02-27 20:23:08,656 INFO [export-for-ncnn.py:349] total parameters: 69920376
2023-02-27 20:23:08,656 INFO [export-for-ncnn.py:351] Using torch.jit.trace()
2023-02-27 20:23:08,656 INFO [export-for-ncnn.py:353] Exporting encoder
2023-02-27 20:23:08,656 INFO [export-for-ncnn.py:174] decode_chunk_len: 32
2023-02-27 20:23:08,656 INFO [export-for-ncnn.py:175] T: 39
/star-fj/fangjun/open-source/icefall-2/egs/librispeech/ASR/pruned_transducer_stateless7_
↪streaming/zipformer2.py:1344: TracerWarning: Converting a tensor to a Python boolean␣
↪might cause the trace to be incorrect. We can't record the data flow of Python values,␣
↪so this value will be treated as a constant in the future. This means that the trace␣
↪might not generalize to other inputs!
  assert cached_len.size(0) == self.num_layers, (
/star-fj/fangjun/open-source/icefall-2/egs/librispeech/ASR/pruned_transducer_stateless7_
↪streaming/zipformer2.py:1348: TracerWarning: Converting a tensor to a Python boolean␣
↪might cause the trace to be incorrect. We can't record the data flow of Python values,␣
↪so this value will be treated as a constant in the future. This means that the trace␣
↪might not generalize to other inputs!
  assert cached_avg.size(0) == self.num_layers, (
/star-fj/fangjun/open-source/icefall-2/egs/librispeech/ASR/pruned_transducer_stateless7_
↪streaming/zipformer2.py:1352: TracerWarning: Converting a tensor to a Python boolean␣
↪might cause the trace to be incorrect. We can't record the data flow of Python values,␣
↪so this value will be treated as a constant in the future. This means that the trace␣
```

(continues on next page)

(continued from previous page)

```
↪might not generalize to other inputs!
  assert cached_key.size(0) == self.num_layers, (
/star-fj/fangjun/open-source/icefall-2/egs/librispeech/ASR/pruned_transducer_stateless7_
↪streaming/zipformer2.py:1356: TracerWarning: Converting a tensor to a Python boolean␣
↪might cause the trace to be incorrect. We can't record the data flow of Python values,␣
↪so this value will be treated as a constant in the future. This means that the trace␣
↪might not generalize to other inputs!
  assert cached_val.size(0) == self.num_layers, (
/star-fj/fangjun/open-source/icefall-2/egs/librispeech/ASR/pruned_transducer_stateless7_
↪streaming/zipformer2.py:1360: TracerWarning: Converting a tensor to a Python boolean␣
↪might cause the trace to be incorrect. We can't record the data flow of Python values,␣
↪so this value will be treated as a constant in the future. This means that the trace␣
↪might not generalize to other inputs!
  assert cached_val2.size(0) == self.num_layers, (
/star-fj/fangjun/open-source/icefall-2/egs/librispeech/ASR/pruned_transducer_stateless7_
↪streaming/zipformer2.py:1364: TracerWarning: Converting a tensor to a Python boolean␣
↪might cause the trace to be incorrect. We can't record the data flow of Python values,␣
↪so this value will be treated as a constant in the future. This means that the trace␣
↪might not generalize to other inputs!
  assert cached_conv1.size(0) == self.num_layers, (
/star-fj/fangjun/open-source/icefall-2/egs/librispeech/ASR/pruned_transducer_stateless7_
↪streaming/zipformer2.py:1368: TracerWarning: Converting a tensor to a Python boolean␣
↪might cause the trace to be incorrect. We can't record the data flow of Python values,␣
↪so this value will be treated as a constant in the future. This means that the trace␣
↪might not generalize to other inputs!
  assert cached_conv2.size(0) == self.num_layers, (
/star-fj/fangjun/open-source/icefall-2/egs/librispeech/ASR/pruned_transducer_stateless7_
↪streaming/zipformer2.py:1373: TracerWarning: Converting a tensor to a Python boolean␣
↪might cause the trace to be incorrect. We can't record the data flow of Python values,␣
↪so this value will be treated as a constant in the future. This means that the trace␣
↪might not generalize to other inputs!
  assert self.left_context_len == cached_key.shape[1], (
/star-fj/fangjun/open-source/icefall-2/egs/librispeech/ASR/pruned_transducer_stateless7_
↪streaming/zipformer2.py:1884: TracerWarning: Converting a tensor to a Python boolean␣
↪might cause the trace to be incorrect. We can't record the data flow of Python values,␣
↪so this value will be treated as a constant in the future. This means that the trace␣
↪might not generalize to other inputs!
  assert self.x_size == x.size(0), (self.x_size, x.size(0))
/star-fj/fangjun/open-source/icefall-2/egs/librispeech/ASR/pruned_transducer_stateless7_
↪streaming/zipformer2.py:2442: TracerWarning: Converting a tensor to a Python boolean␣
↪might cause the trace to be incorrect. We can't record the data flow of Python values,␣
↪so this value will be treated as a constant in the future. This means that the trace␣
↪might not generalize to other inputs!
  assert cached_key.shape[0] == self.left_context_len, (
/star-fj/fangjun/open-source/icefall-2/egs/librispeech/ASR/pruned_transducer_stateless7_
↪streaming/zipformer2.py:2449: TracerWarning: Converting a tensor to a Python boolean␣
↪might cause the trace to be incorrect. We can't record the data flow of Python values,␣
↪so this value will be treated as a constant in the future. This means that the trace␣
↪might not generalize to other inputs!
  assert cached_key.shape[0] == cached_val.shape[0], (
/star-fj/fangjun/open-source/icefall-2/egs/librispeech/ASR/pruned_transducer_stateless7_
↪streaming/zipformer2.py:2469: TracerWarning: Converting a tensor to a Python boolean␣
```

(continues on next page)

(continued from previous page)

```
↪might cause the trace to be incorrect. We can't record the data flow of Python values,␣
↪so this value will be treated as a constant in the future. This means that the trace␣
↪might not generalize to other inputs!
  assert cached_key.shape[0] == left_context_len, (
/star-fj/fangjun/open-source/icefall-2/egs/librispeech/ASR/pruned_transducer_stateless7_
↪streaming/zipformer2.py:2473: TracerWarning: Converting a tensor to a Python boolean␣
↪might cause the trace to be incorrect. We can't record the data flow of Python values,␣
↪so this value will be treated as a constant in the future. This means that the trace␣
↪might not generalize to other inputs!
  assert cached_val.shape[0] == left_context_len, (
/star-fj/fangjun/open-source/icefall-2/egs/librispeech/ASR/pruned_transducer_stateless7_
↪streaming/zipformer2.py:2483: TracerWarning: Converting a tensor to a Python boolean␣
↪might cause the trace to be incorrect. We can't record the data flow of Python values,␣
↪so this value will be treated as a constant in the future. This means that the trace␣
↪might not generalize to other inputs!
  assert kv_len == k.shape[0], (kv_len, k.shape)
/star-fj/fangjun/open-source/icefall-2/egs/librispeech/ASR/pruned_transducer_stateless7_
↪streaming/zipformer2.py:2570: TracerWarning: Converting a tensor to a Python boolean␣
↪might cause the trace to be incorrect. We can't record the data flow of Python values,␣
↪so this value will be treated as a constant in the future. This means that the trace␣
↪might not generalize to other inputs!
  assert list(attn_output.size()) == [bsz * num_heads, seq_len, head_dim // 2]
/star-fj/fangjun/open-source/icefall-2/egs/librispeech/ASR/pruned_transducer_stateless7_
↪streaming/zipformer2.py:2926: TracerWarning: Converting a tensor to a Python boolean␣
↪might cause the trace to be incorrect. We can't record the data flow of Python values,␣
↪so this value will be treated as a constant in the future. This means that the trace␣
↪might not generalize to other inputs!
  assert cache.shape == (x.size(0), x.size(1), self.lorder), (
/star-fj/fangjun/open-source/icefall-2/egs/librispeech/ASR/pruned_transducer_stateless7_
↪streaming/zipformer2.py:2652: TracerWarning: Converting a tensor to a Python boolean␣
↪might cause the trace to be incorrect. We can't record the data flow of Python values,␣
↪so this value will be treated as a constant in the future. This means that the trace␣
↪might not generalize to other inputs!
  assert x.shape[0] == self.x_size, (x.shape[0], self.x_size)
/star-fj/fangjun/open-source/icefall-2/egs/librispeech/ASR/pruned_transducer_stateless7_
↪streaming/zipformer2.py:2653: TracerWarning: Converting a tensor to a Python boolean␣
↪might cause the trace to be incorrect. We can't record the data flow of Python values,␣
↪so this value will be treated as a constant in the future. This means that the trace␣
↪might not generalize to other inputs!
  assert x.shape[2] == self.embed_dim, (x.shape[2], self.embed_dim)
/star-fj/fangjun/open-source/icefall-2/egs/librispeech/ASR/pruned_transducer_stateless7_
↪streaming/zipformer2.py:2666: TracerWarning: Converting a tensor to a Python boolean␣
↪might cause the trace to be incorrect. We can't record the data flow of Python values,␣
↪so this value will be treated as a constant in the future. This means that the trace␣
↪might not generalize to other inputs!
  assert cached_val.shape[0] == self.left_context_len, (
/star-fj/fangjun/open-source/icefall-2/egs/librispeech/ASR/pruned_transducer_stateless7_
↪streaming/zipformer2.py:1543: TracerWarning: Converting a tensor to a Python boolean␣
↪might cause the trace to be incorrect. We can't record the data flow of Python values,␣
↪so this value will be treated as a constant in the future. This means that the trace␣
↪might not generalize to other inputs!
  assert src.shape[0] == self.in_x_size, (src.shape[0], self.in_x_size)
```

(continues on next page)

(continued from previous page)

```
/star-fj/fangjun/open-source/icefall-2/egs/librispeech/ASR/pruned_transducer_stateless7_
↪streaming/zipformer2.py:1637: TracerWarning: Converting a tensor to a Python boolean␣
↪might cause the trace to be incorrect. We can't record the data flow of Python values,␣
↪so this value will be treated as a constant in the future. This means that the trace␣
↪might not generalize to other inputs!
  assert src.shape[0] == self.in_x_size, (
/star-fj/fangjun/open-source/icefall-2/egs/librispeech/ASR/pruned_transducer_stateless7_
↪streaming/zipformer2.py:1643: TracerWarning: Converting a tensor to a Python boolean␣
↪might cause the trace to be incorrect. We can't record the data flow of Python values,␣
↪so this value will be treated as a constant in the future. This means that the trace␣
↪might not generalize to other inputs!
  assert src.shape[2] == self.in_channels, (src.shape[2], self.in_channels)
/star-fj/fangjun/open-source/icefall-2/egs/librispeech/ASR/pruned_transducer_stateless7_
↪streaming/zipformer2.py:1571: TracerWarning: Converting a tensor to a Python boolean␣
↪might cause the trace to be incorrect. We can't record the data flow of Python values,␣
↪so this value will be treated as a constant in the future. This means that the trace␣
↪might not generalize to other inputs!
  if src.shape[0] != self.in_x_size:
/star-fj/fangjun/open-source/icefall-2/egs/librispeech/ASR/pruned_transducer_stateless7_
↪streaming/zipformer2.py:1763: TracerWarning: Converting a tensor to a Python boolean␣
↪might cause the trace to be incorrect. We can't record the data flow of Python values,␣
↪so this value will be treated as a constant in the future. This means that the trace␣
↪might not generalize to other inputs!
  assert src1.shape[:-1] == src2.shape[:-1], (src1.shape, src2.shape)
/star-fj/fangjun/open-source/icefall-2/egs/librispeech/ASR/pruned_transducer_stateless7_
↪streaming/zipformer2.py:1779: TracerWarning: Converting a tensor to a Python boolean␣
↪might cause the trace to be incorrect. We can't record the data flow of Python values,␣
↪so this value will be treated as a constant in the future. This means that the trace␣
↪might not generalize to other inputs!
  assert src1.shape[-1] == self.dim1, (src1.shape[-1], self.dim1)
/star-fj/fangjun/open-source/icefall-2/egs/librispeech/ASR/pruned_transducer_stateless7_
↪streaming/zipformer2.py:1780: TracerWarning: Converting a tensor to a Python boolean␣
↪might cause the trace to be incorrect. We can't record the data flow of Python values,␣
↪so this value will be treated as a constant in the future. This means that the trace␣
↪might not generalize to other inputs!
  assert src2.shape[-1] == self.dim2, (src2.shape[-1], self.dim2)
/star-fj/fangjun/py38/lib/python3.8/site-packages/torch/jit/_trace.py:958:␣
↪TracerWarning: Encountering a list at the output of the tracer might cause the trace␣
↪to be incorrect, this is only valid if the container structure does not change based␣
↪on the module's inputs. Consider using a constant container instead (e.g. for `list`,␣
↪use a `tuple` instead. for `dict`, use a `NamedTuple` instead). If you absolutely need␣
↪this and know the side effects, pass strict=False to trace() to allow this behavior.
  module._c._create_method_from_trace(
2023-02-27 20:23:19,640 INFO [export-for-ncnn.py:182] Saved to icefall-asr-librispeech-
↪pruned-transducer-stateless7-streaming-2022-12-29/exp/encoder_jit_trace-pnnx.pt
2023-02-27 20:23:19,646 INFO [export-for-ncnn.py:357] Exporting decoder
/star-fj/fangjun/open-source/icefall-2/egs/librispeech/ASR/pruned_transducer_stateless7_
↪streaming/decoder.py:102: TracerWarning: Converting a tensor to a Python boolean might␣
↪cause the trace to be incorrect. We can't record the data flow of Python values, so␣
↪this value will be treated as a constant in the future. This means that the trace␣
↪might not generalize to other inputs!
  assert embedding_out.size(-1) == self.context_size
```

(continues on next page)

(continued from previous page)

```
2023-02-27 20:23:19,686 INFO [export-for-ncnn.py:204] Saved to icefall-asr-librispeech-
↪pruned-transducer-stateless7-streaming-2022-12-29/exp/decoder_jit_trace-pnnx.pt
2023-02-27 20:23:19,686 INFO [export-for-ncnn.py:361] Exporting joiner
2023-02-27 20:23:19,735 INFO [export-for-ncnn.py:231] Saved to icefall-asr-librispeech-
↪pruned-transducer-stateless7-streaming-2022-12-29/exp/joiner_jit_trace-pnnx.pt
```

The log shows the model has `69920376` parameters, i.e., `~69.9 M`.

```
ls -lh icefall-asr-librispeech-pruned-transducer-stateless7-streaming-2022-12-29/exp/
↪pretrained.pt
-rw-r--r-- 1 kuangfangjun root 269M Jan 12 12:53 icefall-asr-librispeech-pruned-
↪transducer-stateless7-streaming-2022-12-29/exp/pretrained.pt
```

You can see that the file size of the pre-trained model is `269 MB`, which is roughly equal to `69920376*4/1024/1024 = 266.725 MB`.

After running `pruned_transducer_stateless7_streaming/export-for-ncnn.py`, we will get the following files:

```
ls -lh icefall-asr-librispeech-pruned-transducer-stateless7-streaming-2022-12-29/exp/
↪*pnnx.pt

-rw-r--r-- 1 kuangfangjun root 1022K Feb 27 20:23 icefall-asr-librispeech-pruned-
↪transducer-stateless7-streaming-2022-12-29/exp/decoder_jit_trace-pnnx.pt
-rw-r--r-- 1 kuangfangjun root  266M Feb 27 20:23 icefall-asr-librispeech-pruned-
↪transducer-stateless7-streaming-2022-12-29/exp/encoder_jit_trace-pnnx.pt
-rw-r--r-- 1 kuangfangjun root  2.8M Feb 27 20:23 icefall-asr-librispeech-pruned-
↪transducer-stateless7-streaming-2022-12-29/exp/joiner_jit_trace-pnnx.pt
```

4. Export torchscript model via pnnx

Hint: Make sure you have set up the `PATH` environment variable in *2. Install ncnn and pnnx*. Otherwise, it will throw an error saying that `pnnx` could not be found.

Now, it's time to export our models to ncnn via `pnnx`.

```
cd icefall-asr-librispeech-pruned-transducer-stateless7-streaming-2022-12-29/exp/

pnnx ./encoder_jit_trace-pnnx.pt
pnnx ./decoder_jit_trace-pnnx.pt
pnnx ./joiner_jit_trace-pnnx.pt
```

It will generate the following files:

```
ls -lh  icefall-asr-librispeech-pruned-transducer-stateless7-streaming-2022-12-29/exp/
↪*ncnn*{bin,param}

-rw-r--r-- 1 kuangfangjun root 509K Feb 27 20:31 icefall-asr-librispeech-pruned-
↪transducer-stateless7-streaming-2022-12-29/exp/decoder_jit_trace-pnnx.ncnn.bin
-rw-r--r-- 1 kuangfangjun root  437 Feb 27 20:31 icefall-asr-librispeech-pruned-
```

(continues on next page)

(continued from previous page)

```
↪transducer-stateless7-streaming-2022-12-29/exp/decoder_jit_trace-pnnx.ncnn.param
-rw-r--r-- 1 kuangfangjun root 133M Feb 27 20:30 icefall-asr-librispeech-pruned-
↪transducer-stateless7-streaming-2022-12-29/exp/encoder_jit_trace-pnnx.ncnn.bin
-rw-r--r-- 1 kuangfangjun root 152K Feb 27 20:30 icefall-asr-librispeech-pruned-
↪transducer-stateless7-streaming-2022-12-29/exp/encoder_jit_trace-pnnx.ncnn.param
-rw-r--r-- 1 kuangfangjun root 1.4M Feb 27 20:31 icefall-asr-librispeech-pruned-
↪transducer-stateless7-streaming-2022-12-29/exp/joiner_jit_trace-pnnx.ncnn.bin
-rw-r--r-- 1 kuangfangjun root  488 Feb 27 20:31 icefall-asr-librispeech-pruned-
↪transducer-stateless7-streaming-2022-12-29/exp/joiner_jit_trace-pnnx.ncnn.param
```

There are two types of files:

- `param`: It is a text file containing the model architectures. You can use a text editor to view its content.
- `bin`: It is a binary file containing the model parameters.

We compare the file sizes of the models below before and after converting via `pnnx`:

File name	File size
encoder_jit_trace-pnnx.pt	266 MB
decoder_jit_trace-pnnx.pt	1022 KB
joiner_jit_trace-pnnx.pt	2.8 MB
encoder_jit_trace-pnnx.ncnn.bin	133 MB
decoder_jit_trace-pnnx.ncnn.bin	509 KB
joiner_jit_trace-pnnx.ncnn.bin	1.4 MB

You can see that the file sizes of the models after conversion are about one half of the models before conversion:

- encoder: 266 MB vs 133 MB
- decoder: 1022 KB vs 509 KB
- joiner: 2.8 MB vs 1.4 MB

The reason is that by default `pnnx` converts `float32` parameters to `float16`. A `float32` parameter occupies 4 bytes, while it is 2 bytes for `float16`. Thus, it is `twice smaller` after conversion.

Hint: If you use `pnnx ./encoder_jit_trace-pnnx.pt fp16=0`, then `pnnx` won't convert `float32` to `float16`.

5. Test the exported models in icefall

Note: We assume you have set up the environment variable `PYTHONPATH` when building ncnn.

Now we have successfully converted our pre-trained model to ncnn format. The generated 6 files are what we need. You can use the following code to test the converted models:

```
python3 ./pruned_transducer_stateless7_streaming/streaming-ncnn-decode.py \
  --tokens ./icefall-asr-librispeech-pruned-transducer-stateless7-streaming-2022-12-29/
↪data/lang_bpe_500/tokens.txt \
  --encoder-param-filename ./icefall-asr-librispeech-pruned-transducer-stateless7-
↪streaming-2022-12-29/exp/encoder_jit_trace-pnnx.ncnn.param \
```

(continues on next page)

(continued from previous page)

```
  --encoder-bin-filename ./icefall-asr-librispeech-pruned-transducer-stateless7-
↪streaming-2022-12-29/exp/encoder_jit_trace-pnnx.ncnn.bin \
  --decoder-param-filename ./icefall-asr-librispeech-pruned-transducer-stateless7-
↪streaming-2022-12-29/exp/decoder_jit_trace-pnnx.ncnn.param \
  --decoder-bin-filename ./icefall-asr-librispeech-pruned-transducer-stateless7-
↪streaming-2022-12-29/exp/decoder_jit_trace-pnnx.ncnn.bin \
  --joiner-param-filename ./icefall-asr-librispeech-pruned-transducer-stateless7-
↪streaming-2022-12-29/exp/joiner_jit_trace-pnnx.ncnn.param \
  --joiner-bin-filename ./icefall-asr-librispeech-pruned-transducer-stateless7-streaming-
↪2022-12-29/exp/joiner_jit_trace-pnnx.ncnn.bin \
  ./icefall-asr-librispeech-pruned-transducer-stateless7-streaming-2022-12-29/test_wavs/
↪1089-134686-0001.wav
```

Hint: ncnn supports only `batch size == 1`, so `streaming-ncnn-decode.py` accepts only 1 wave file as input.

The output is given below:

```
2023-02-27 20:43:40,283 INFO [streaming-ncnn-decode.py:349] {'tokens': './icefall-asr-
↪librispeech-pruned-transducer-stateless7-streaming-2022-12-29/data/lang_bpe_500/tokens.
↪txt', 'encoder_param_filename': './icefall-asr-librispeech-pruned-transducer-
↪stateless7-streaming-2022-12-29/exp/encoder_jit_trace-pnnx.ncnn.param', 'encoder_bin_
↪filename': './icefall-asr-librispeech-pruned-transducer-stateless7-streaming-2022-12-
↪29/exp/encoder_jit_trace-pnnx.ncnn.bin', 'decoder_param_filename': './icefall-asr-
↪librispeech-pruned-transducer-stateless7-streaming-2022-12-29/exp/decoder_jit_trace-
↪pnnx.ncnn.param', 'decoder_bin_filename': './icefall-asr-librispeech-pruned-transducer-
↪stateless7-streaming-2022-12-29/exp/decoder_jit_trace-pnnx.ncnn.bin', 'joiner_param_
↪filename': './icefall-asr-librispeech-pruned-transducer-stateless7-streaming-2022-12-
↪29/exp/joiner_jit_trace-pnnx.ncnn.param', 'joiner_bin_filename': './icefall-asr-
↪librispeech-pruned-transducer-stateless7-streaming-2022-12-29/exp/joiner_jit_trace-
↪pnnx.ncnn.bin', 'sound_filename': './icefall-asr-librispeech-pruned-transducer-
↪stateless7-streaming-2022-12-29/test_wavs/1089-134686-0001.wav'}
2023-02-27 20:43:41,260 INFO [streaming-ncnn-decode.py:357] Constructing Fbank computer
2023-02-27 20:43:41,264 INFO [streaming-ncnn-decode.py:360] Reading sound files: ./
↪icefall-asr-librispeech-pruned-transducer-stateless7-streaming-2022-12-29/test_wavs/
↪1089-134686-0001.wav
2023-02-27 20:43:41,269 INFO [streaming-ncnn-decode.py:365] torch.Size([106000])
2023-02-27 20:43:41,280 INFO [streaming-ncnn-decode.py:372] number of states: 35
2023-02-27 20:43:45,026 INFO [streaming-ncnn-decode.py:410] ./icefall-asr-librispeech-
↪pruned-transducer-stateless7-streaming-2022-12-29/test_wavs/1089-134686-0001.wav
2023-02-27 20:43:45,026 INFO [streaming-ncnn-decode.py:411] AFTER EARLY NIGHTFALL THE␣
↪YELLOW LAMPS WOULD LIGHT UP HERE AND THERE THE SQUALID QUARTER OF THE BROTHELS
```

Congratulations! You have successfully exported a model from PyTorch to ncnn!

6. Modify the exported encoder for sherpa-ncnn

In order to use the exported models in sherpa-ncnn, we have to modify `encoder_jit_trace-pnnx.ncnn.param`.

Let us have a look at the first few lines of `encoder_jit_trace-pnnx.ncnn.param`:

```
7767517
2028 2547
Input                    in0                      0 1 in0
```

Explanation of the above three lines:

1. `7767517`, it is a magic number and should not be changed.
2. `2028 2547`, the first number `2028` specifies the number of layers in this file, while `2547` specifies the number of intermediate outputs of this file
3. `Input in0 0 1 in0`, `Input` is the layer type of this layer; `in0` is the layer name of this layer; `0` means this layer has no input; `1` means this layer has one output; `in0` is the output name of this layer.

We need to add 1 extra line and also increment the number of layers. The result looks like below:

```
7767517
2029 2547
SherpaMetaData           sherpa_meta_data1        0 0 0=2 1=32 2=4 3=7 15=1 -23316=5,2,4,
↪3,2,4 -23317=5,384,384,384,384,384 -23318=5,192,192,192,192,192 -23319=5,1,2,4,8,2 -
↪23320=5,31,31,31,31,31
Input                    in0                      0 1 in0
```

Explanation

1. `7767517`, it is still the same
2. `2029 2547`, we have added an extra layer, so we need to update `2028` to `2029`. We don't need to change `2547` since the newly added layer has no inputs or outputs.
3. `SherpaMetaData sherpa_meta_data1 0 0 0=2 1=32 2=4 3=7 -23316=5,2,4,3,2,4 -23317=5, 384,384,384,384,384 -23318=5,192,192,192,192,192 -23319=5,1,2,4,8,2 -23320=5,31,31, 31,31,31` This line is newly added. Its explanation is given below:
   - `SherpaMetaData` is the type of this layer. Must be `SherpaMetaData`.
   - `sherpa_meta_data1` is the name of this layer. Must be `sherpa_meta_data1`.
   - `0 0` means this layer has no inputs or output. Must be `0 0`
   - `0=2`, 0 is the key and 2 is the value. MUST be `0=2`
   - `1=32`, 1 is the key and 32 is the value of the parameter `--decode-chunk-len` that you provided when running `./pruned_transducer_stateless7_streaming/export-for-ncnn.py`.
   - `2=4`, 2 is the key and 4 is the value of the parameter `--num-left-chunks` that you provided when running `./pruned_transducer_stateless7_streaming/export-for-ncnn.py`.
   - `3=7`, 3 is the key and 7 is the value of for the amount of padding used in the Conv2DSubsampling layer. It should be 7 for zipformer if you don't change zipformer.py.
   - `15=1`, attribute 15, this is the model version. Starting from sherpa-ncnn v2.0, we require that the model version has to be >= 1.
   - `-23316=5,2,4,3,2,4`, attribute 16, this is an array attribute. It is attribute 16 since -23300 - (-23316) = 16. The first element of the array is the length of the array, which is 5 in our case.

2,4,3,2,4 is the value of `--num-encoder-layers``that you provided when running ``./pruned_transducer_stateless7_streaming/export-for-ncnn.py`.

- `-23317=5,384,384,384,384,384`, attribute 17. The first element of the array is the length of the array, which is 5 in our case. `384,384,384,384,384` is the value of `--encoder-dims``that you provided when running ``./pruned_transducer_stateless7_streaming/export-for-ncnn.py`.
- `-23318=5,192,192,192,192,192`, attribute 18. The first element of the array is the length of the array, which is 5 in our case. `192,192,192,192,192` is the value of `--attention-dims` that you provided when running `./pruned_transducer_stateless7_streaming/export-for-ncnn.py`.
- `-23319=5,1,2,4,8,2`, attribute 19. The first element of the array is the length of the array, which is 5 in our case. `1,2,4,8,2` is the value of `--zipformer-downsampling-factors` that you provided when running `./pruned_transducer_stateless7_streaming/export-for-ncnn.py`.
- `-23320=5,31,31,31,31,31`, attribute 20. The first element of the array is the length of the array, which is 5 in our case. `31,31,31,31,31` is the value of `--cnn-module-kernels` that you provided when running `./pruned_transducer_stateless7_streaming/export-for-ncnn.py`.

For ease of reference, we list the key-value pairs that you need to add in the following table. If your model has a different setting, please change the values for `SherpaMetaData` accordingly. Otherwise, you will be `SAD`.

key	value
0	2 (fixed)
1	`-decode-chunk-len`
2	`--num-left-chunks`
3	7 (if you don't change code)
15	1 (The model version)
-23316	`--num-encoder-layer`
-23317	`--encoder-dims`
-23318	`--attention-dims`
-23319	`--zipformer-downsampling-factors`
-23320	`--cnn-module-kernels`

4. `Input in0 0 1 in0`. No need to change it.

Caution: When you add a new layer `SherpaMetaData`, please remember to update the number of layers. In our case, update `2028` to `2029`. Otherwise, you will be SAD later.

Hint: After adding the new layer `SherpaMetaData`, you cannot use this model with `streaming-ncnn-decode.py` anymore since `SherpaMetaData` is supported only in sherpa-ncnn.

Hint: ncnn is very flexible. You can add new layers to it just by text-editing the `param` file! You don't need to change the `bin` file.

Now you can use this model in sherpa-ncnn. Please refer to the following documentation:

- Linux/macOS/Windows/arm/aarch64: https://k2-fsa.github.io/sherpa/ncnn/install/index.html
- Android: https://k2-fsa.github.io/sherpa/ncnn/android/index.html
- iOS: https://k2-fsa.github.io/sherpa/ncnn/ios/index.html
- Python: https://k2-fsa.github.io/sherpa/ncnn/python/index.html

We have a list of pre-trained models that have been exported for sherpa-ncnn:

- https://k2-fsa.github.io/sherpa/ncnn/pretrained_models/index.html

  You can find more usages there.

5.5.2 Export ConvEmformer transducer models to ncnn

We use the pre-trained model from the following repository as an example:

- https://huggingface.co/Zengwei/icefall-asr-librispeech-conv-emformer-transducer-stateless2-2022-07-05

We will show you step by step how to export it to ncnn and run it with sherpa-ncnn.

Hint: We use `Ubuntu 18.04`, `torch 1.13`, and `Python 3.8` for testing.

Caution: Please use a more recent version of PyTorch. For instance, `torch 1.8` may `not` work.

1. Download the pre-trained model

Hint: You can also refer to https://k2-fsa.github.io/sherpa/cpp/pretrained_models/online_transducer.html#icefall-asr-librispeech-conv-emformer-transducer-stateless2-2022-07-05 to download the pre-trained model.

You have to install git-lfs before you continue.

```
cd egs/librispeech/ASR

GIT_LFS_SKIP_SMUDGE=1 git clone https://huggingface.co/Zengwei/icefall-asr-librispeech-
↪conv-emformer-transducer-stateless2-2022-07-05
cd icefall-asr-librispeech-conv-emformer-transducer-stateless2-2022-07-05

git lfs pull --include "exp/pretrained-epoch-30-avg-10-averaged.pt"
git lfs pull --include "data/lang_bpe_500/bpe.model"

cd ..
```

Note: We downloaded `exp/pretrained-xxx.pt`, not `exp/cpu-jit_xxx.pt`.

In the above code, we downloaded the pre-trained model into the directory `egs/librispeech/ASR/icefall-asr-librispeech-conv-emformer-transducer-stateless2-2022-07-05`.

2. Install ncnn and pnnx

```
# We put ncnn into $HOME/open-source/ncnn
# You can change it to anywhere you like

cd $HOME
mkdir -p open-source
cd open-source

git clone https://github.com/csukuangfj/ncnn
cd ncnn
git submodule update --recursive --init

# Note: We don't use "python setup.py install" or "pip install ." here

mkdir -p build-wheel
cd build-wheel

cmake \
  -DCMAKE_BUILD_TYPE=Release \
  -DNCNN_PYTHON=ON \
  -DNCNN_BUILD_BENCHMARK=OFF \
  -DNCNN_BUILD_EXAMPLES=OFF \
  -DNCNN_BUILD_TOOLS=ON \
..

make -j4

cd ..

# Note: $PWD here is $HOME/open-source/ncnn

export PYTHONPATH=$PWD/python:$PYTHONPATH
export PATH=$PWD/tools/pnnx/build/src:$PATH
export PATH=$PWD/build-wheel/tools/quantize:$PATH

# Now build pnnx
cd tools/pnnx
mkdir build
cd build
cmake ..
make -j4

./src/pnnx
```

Congratulations! You have successfully installed the following components:

- `pnnx`, which is an executable located in `$HOME/open-source/ncnn/tools/pnnx/build/src`. We will use it to convert models exported by `torch.jit.trace()`.
- `ncnn2int8`, which is an executable located in `$HOME/open-source/ncnn/build-wheel/tools/quantize`. We will use it to quantize our models to `int8`.
- `ncnn.cpython-38-x86_64-linux-gnu.so`, which is a Python module located in `$HOME/open-source/ncnn/python/ncnn`.

Note: I am using `Python 3.8`, so it is `ncnn.cpython-38-x86_64-linux-gnu.so`. If you use a different version, say, `Python 3.9`, the name would be `ncnn.cpython-39-x86_64-linux-gnu.so`.

Also, if you are not using Linux, the file name would also be different. But that does not matter. As long as you can compile it, it should work.

We have set up `PYTHONPATH` so that you can use `import ncnn` in your Python code. We have also set up `PATH` so that you can use `pnnx` and `ncnn2int8` later in your terminal.

Caution: Please don't use https://github.com/tencent/ncnn. We have made some modifications to the offical ncnn.

We will synchronize https://github.com/csukuangfj/ncnn periodically with the official one.

3. Export the model via torch.jit.trace()

First, let us rename our pre-trained model:

```
cd egs/librispeech/ASR

cd icefall-asr-librispeech-conv-emformer-transducer-stateless2-2022-07-05/exp

ln -s pretrained-epoch-30-avg-10-averaged.pt epoch-30.pt

cd ../..
```

Next, we use the following code to export our model:

```
dir=./icefall-asr-librispeech-conv-emformer-transducer-stateless2-2022-07-05/

./conv_emformer_transducer_stateless2/export-for-ncnn.py \
  --exp-dir $dir/exp \
  --tokens $dir/data/lang_bpe_500/tokens.txt \
  --epoch 30 \
  --avg 1 \
  --use-averaged-model 0 \
  --num-encoder-layers 12 \
  --chunk-length 32 \
  --cnn-module-kernel 31 \
  --left-context-length 32 \
  --right-context-length 8 \
  --memory-size 32 \
  --encoder-dim 512
```

Caution: If your model has different configuration parameters, please change them accordingly.

Hint: We have renamed our model to `epoch-30.pt` so that we can use `--epoch 30`. There is only one pre-trained model, so we use `--avg 1 --use-averaged-model 0`.

If you have trained a model by yourself and if you have all checkpoints available, please first use `decode.py` to tune `--epoch --avg` and select the best combination with with `--use-averaged-model 1`.

Note: You will see the following log output:

```
2023-01-11 12:15:38,677 INFO [export-for-ncnn.py:220] device: cpu
2023-01-11 12:15:38,681 INFO [export-for-ncnn.py:229] {'best_train_loss': inf, 'best_
↪valid_loss': inf, 'best_train_epoch': -1, 'best_v
alid_epoch': -1, 'batch_idx_train': 0, 'log_interval': 50, 'reset_interval': 200, 'valid_
↪interval': 3000, 'feature_dim': 80, 'subsampl
ing_factor': 4, 'decoder_dim': 512, 'joiner_dim': 512, 'model_warm_step': 3000, 'env_info
↪': {'k2-version': '1.23.2', 'k2-build-type':
'Release', 'k2-with-cuda': True, 'k2-git-sha1': 'a34171ed85605b0926eebbd0463d059431f4f74a
↪', 'k2-git-date': 'Wed Dec 14 00:06:38 2022',
 'lhotse-version': '1.12.0.dev+missing.version.file', 'torch-version': '1.10.0+cu102',
↪'torch-cuda-available': False, 'torch-cuda-vers
ion': '10.2', 'python-version': '3.8', 'icefall-git-branch': 'fix-stateless3-train-2022-
↪12-27', 'icefall-git-sha1': '530e8a1-dirty', '
icefall-git-date': 'Tue Dec 27 13:59:18 2022', 'icefall-path': '/star-fj/fangjun/open-
↪source/icefall', 'k2-path': '/star-fj/fangjun/op
en-source/k2/k2/python/k2/__init__.py', 'lhotse-path': '/star-fj/fangjun/open-source/
↪lhotse/lhotse/__init__.py', 'hostname': 'de-74279
-k2-train-3-1220120619-7695ff496b-s9n4w', 'IP address': '127.0.0.1'}, 'epoch': 30, 'iter
↪': 0, 'avg': 1, 'exp_dir': PosixPath('icefa
ll-asr-librispeech-conv-emformer-transducer-stateless2-2022-07-05/exp'), 'bpe_model': './
↪icefall-asr-librispeech-conv-emformer-transdu
cer-stateless2-2022-07-05//data/lang_bpe_500/bpe.model', 'jit': False, 'context_size': 2,
↪ 'use_averaged_model': False, 'encoder_dim':
512, 'nhead': 8, 'dim_feedforward': 2048, 'num_encoder_layers': 12, 'cnn_module_kernel':␣
↪31, 'left_context_length': 32, 'chunk_length'
: 32, 'right_context_length': 8, 'memory_size': 32, 'blank_id': 0, 'vocab_size': 500}
2023-01-11 12:15:38,681 INFO [export-for-ncnn.py:231] About to create model
2023-01-11 12:15:40,053 INFO [checkpoint.py:112] Loading checkpoint from icefall-asr-
↪librispeech-conv-emformer-transducer-stateless2-2
022-07-05/exp/epoch-30.pt
2023-01-11 12:15:40,708 INFO [export-for-ncnn.py:315] Number of model parameters:␣
↪75490012
2023-01-11 12:15:41,681 INFO [export-for-ncnn.py:318] Using torch.jit.trace()
2023-01-11 12:15:41,681 INFO [export-for-ncnn.py:320] Exporting encoder
2023-01-11 12:15:41,682 INFO [export-for-ncnn.py:149] chunk_length: 32, right_context_
↪length: 8
```

The log shows the model has `75490012` parameters, i.e., `~75 M`.

```
ls -lh icefall-asr-librispeech-conv-emformer-transducer-stateless2-2022-07-05/exp/
↪pretrained-epoch-30-avg-10-averaged.pt

-rw-r--r-- 1 kuangfangjun root 289M Jan 11 12:05 icefall-asr-librispeech-conv-emformer-
↪transducer-stateless2-2022-07-05/exp/pretrained-epoch-30-avg-10-averaged.pt
```

You can see that the file size of the pre-trained model is `289 MB`, which is roughly equal to `75490012*4/1024/1024 = 287.97 MB`.

After running conv_emformer_transducer_stateless2/export-for-ncnn.py, we will get the following files:

```
ls -lh icefall-asr-librispeech-conv-emformer-transducer-stateless2-2022-07-05/exp/*pnnx*

-rw-r--r-- 1 kuangfangjun root 1010K Jan 11 12:15 icefall-asr-librispeech-conv-emformer-
↪transducer-stateless2-2022-07-05/exp/decoder_jit_trace-pnnx.pt
-rw-r--r-- 1 kuangfangjun root  283M Jan 11 12:15 icefall-asr-librispeech-conv-emformer-
↪transducer-stateless2-2022-07-05/exp/encoder_jit_trace-pnnx.pt
-rw-r--r-- 1 kuangfangjun root  3.0M Jan 11 12:15 icefall-asr-librispeech-conv-emformer-
↪transducer-stateless2-2022-07-05/exp/joiner_jit_trace-pnnx.pt
```

4. Export torchscript model via pnnx

Hint: Make sure you have set up the PATH environment variable. Otherwise, it will throw an error saying that pnnx could not be found.

Now, it's time to export our models to ncnn via pnnx.

```
cd icefall-asr-librispeech-conv-emformer-transducer-stateless2-2022-07-05/exp/

pnnx ./encoder_jit_trace-pnnx.pt
pnnx ./decoder_jit_trace-pnnx.pt
pnnx ./joiner_jit_trace-pnnx.pt
```

It will generate the following files:

```
ls -lh  icefall-asr-librispeech-conv-emformer-transducer-stateless2-2022-07-05/exp/*ncnn*
↪{bin,param}

-rw-r--r-- 1 kuangfangjun root 503K Jan 11 12:38 icefall-asr-librispeech-conv-emformer-
↪transducer-stateless2-2022-07-05/exp/decoder_jit_trace-pnnx.ncnn.bin
-rw-r--r-- 1 kuangfangjun root  437 Jan 11 12:38 icefall-asr-librispeech-conv-emformer-
↪transducer-stateless2-2022-07-05/exp/decoder_jit_trace-pnnx.ncnn.param
-rw-r--r-- 1 kuangfangjun root 142M Jan 11 12:36 icefall-asr-librispeech-conv-emformer-
↪transducer-stateless2-2022-07-05/exp/encoder_jit_trace-pnnx.ncnn.bin
-rw-r--r-- 1 kuangfangjun root  79K Jan 11 12:36 icefall-asr-librispeech-conv-emformer-
↪transducer-stateless2-2022-07-05/exp/encoder_jit_trace-pnnx.ncnn.param
-rw-r--r-- 1 kuangfangjun root 1.5M Jan 11 12:38 icefall-asr-librispeech-conv-emformer-
↪transducer-stateless2-2022-07-05/exp/joiner_jit_trace-pnnx.ncnn.bin
-rw-r--r-- 1 kuangfangjun root  488 Jan 11 12:38 icefall-asr-librispeech-conv-emformer-
↪transducer-stateless2-2022-07-05/exp/joiner_jit_trace-pnnx.ncnn.param
```

There are two types of files:

- param: It is a text file containing the model architectures. You can use a text editor to view its content.
- bin: It is a binary file containing the model parameters.

We compare the file sizes of the models below before and after converting via pnnx:

File name	File size
encoder_jit_trace-pnnx.pt	283 MB
decoder_jit_trace-pnnx.pt	1010 KB
joiner_jit_trace-pnnx.pt	3.0 MB
encoder_jit_trace-pnnx.ncnn.bin	142 MB
decoder_jit_trace-pnnx.ncnn.bin	503 KB
joiner_jit_trace-pnnx.ncnn.bin	1.5 MB

You can see that the file sizes of the models after conversion are about one half of the models before conversion:

- encoder: 283 MB vs 142 MB
- decoder: 1010 KB vs 503 KB
- joiner: 3.0 MB vs 1.5 MB

The reason is that by default `pnnx` converts `float32` parameters to `float16`. A `float32` parameter occupies 4 bytes, while it is 2 bytes for `float16`. Thus, it is `twice smaller` after conversion.

Hint: If you use `pnnx ./encoder_jit_trace-pnnx.pt fp16=0`, then `pnnx` won't convert `float32` to `float16`.

5. Test the exported models in icefall

Note: We assume you have set up the environment variable `PYTHONPATH` when building ncnn.

Now we have successfully converted our pre-trained model to ncnn format. The generated 6 files are what we need. You can use the following code to test the converted models:

```
./conv_emformer_transducer_stateless2/streaming-ncnn-decode.py \
  --tokens ./icefall-asr-librispeech-conv-emformer-transducer-stateless2-2022-07-05/data/
↪lang_bpe_500/tokens.txt \
  --encoder-param-filename ./icefall-asr-librispeech-conv-emformer-transducer-stateless2-
↪2022-07-05/exp/encoder_jit_trace-pnnx.ncnn.param \
  --encoder-bin-filename ./icefall-asr-librispeech-conv-emformer-transducer-stateless2-
↪2022-07-05/exp/encoder_jit_trace-pnnx.ncnn.bin \
  --decoder-param-filename ./icefall-asr-librispeech-conv-emformer-transducer-stateless2-
↪2022-07-05/exp/decoder_jit_trace-pnnx.ncnn.param \
  --decoder-bin-filename ./icefall-asr-librispeech-conv-emformer-transducer-stateless2-
↪2022-07-05/exp/decoder_jit_trace-pnnx.ncnn.bin \
  --joiner-param-filename ./icefall-asr-librispeech-conv-emformer-transducer-stateless2-
↪2022-07-05/exp/joiner_jit_trace-pnnx.ncnn.param \
  --joiner-bin-filename ./icefall-asr-librispeech-conv-emformer-transducer-stateless2-
↪2022-07-05/exp/joiner_jit_trace-pnnx.ncnn.bin \
  ./icefall-asr-librispeech-conv-emformer-transducer-stateless2-2022-07-05/test_wavs/
↪1089-134686-0001.wav
```

Hint: ncnn supports only `batch size == 1`, so `streaming-ncnn-decode.py` accepts only 1 wave file as input.

The output is given below:

```
2023-01-11 14:02:12,216 INFO [streaming-ncnn-decode.py:320] {'tokens': './icefall-asr-
↪librispeech-conv-emformer-transducer-stateless2-2022-07-05/data/lang_bpe_500/tokens.txt
↪', 'encoder_param_filename': './icefall-asr-librispeech-conv-emformer-transducer-
↪stateless2-2022-07-05/exp/encoder_jit_trace-pnnx.ncnn.param', 'encoder_bin_filename':
↪'./icefall-asr-librispeech-conv-emformer-transducer-stateless2-2022-07-05/exp/encoder_
↪jit_trace-pnnx.ncnn.bin', 'decoder_param_filename': './icefall-asr-librispeech-conv-
↪emformer-transducer-stateless2-2022-07-05/exp/decoder_jit_trace-pnnx.ncnn.param',
↪'decoder_bin_filename': './icefall-asr-librispeech-conv-emformer-transducer-stateless2-
↪2022-07-05/exp/decoder_jit_trace-pnnx.ncnn.bin', 'joiner_param_filename': './icefall-
↪asr-librispeech-conv-emformer-transducer-stateless2-2022-07-05/exp/joiner_jit_trace-
↪pnnx.ncnn.param', 'joiner_bin_filename': './icefall-asr-librispeech-conv-emformer-
↪transducer-stateless2-2022-07-05/exp/joiner_jit_trace-pnnx.ncnn.bin', 'sound_filename
↪': './icefall-asr-librispeech-conv-emformer-transducer-stateless2-2022-07-05/test_wavs/
↪1089-134686-0001.wav'}
T 51 32
2023-01-11 14:02:13,141 INFO [streaming-ncnn-decode.py:328] Constructing Fbank computer
2023-01-11 14:02:13,151 INFO [streaming-ncnn-decode.py:331] Reading sound files: ./
↪icefall-asr-librispeech-conv-emformer-transducer-stateless2-2022-07-05/test_wavs/1089-
↪134686-0001.wav
2023-01-11 14:02:13,176 INFO [streaming-ncnn-decode.py:336] torch.Size([106000])
2023-01-11 14:02:17,581 INFO [streaming-ncnn-decode.py:380] ./icefall-asr-librispeech-
↪conv-emformer-transducer-stateless2-2022-07-05/test_wavs/1089-134686-0001.wav
2023-01-11 14:02:17,581 INFO [streaming-ncnn-decode.py:381] AFTER EARLY NIGHTFALL THE␣
↪YELLOW LAMPS WOULD LIGHT UP HERE AND THERE THE SQUALID QUARTER OF THE BROTHELS
```

Congratulations! You have successfully exported a model from PyTorch to ncnn!

6. Modify the exported encoder for sherpa-ncnn

In order to use the exported models in sherpa-ncnn, we have to modify `encoder_jit_trace-pnnx.ncnn.param`.

Let us have a look at the first few lines of `encoder_jit_trace-pnnx.ncnn.param`:

```
7767517
1060 1342
Input                    in0                      0 1 in0
```

Explanation of the above three lines:

1. `7767517`, it is a magic number and should not be changed.
2. `1060 1342`, the first number `1060` specifies the number of layers in this file, while `1342` specifies the number of intermediate outputs of this file
3. `Input in0 0 1 in0`, `Input` is the layer type of this layer; `in0` is the layer name of this layer; `0` means this layer has no input; `1` means this layer has one output; `in0` is the output name of this layer.

We need to add 1 extra line and also increment the number of layers. The result looks like below:

```
7767517
1061 1342
SherpaMetaData           sherpa_meta_data1        0 0 0=1 1=12 2=32 3=31 4=8 5=32 6=8␣
↪7=512
Input                    in0                      0 1 in0
```

Explanation

1. `7767517`, it is still the same
2. `1061 1342`, we have added an extra layer, so we need to update `1060` to `1061`. We don't need to change `1342` since the newly added layer has no inputs or outputs.
3. `SherpaMetaData sherpa_meta_data1 0 0 0=1 1=12 2=32 3=31 4=8 5=32 6=8 7=512` This line is newly added. Its explanation is given below:
   - `SherpaMetaData` is the type of this layer. Must be `SherpaMetaData`.
   - `sherpa_meta_data1` is the name of this layer. Must be `sherpa_meta_data1`.
   - `0 0` means this layer has no inputs or output. Must be `0 0`
   - `0=1`, 0 is the key and 1 is the value. MUST be `0=1`
   - `1=12`, 1 is the key and 12 is the value of the parameter `--num-encoder-layers` that you provided when running `conv_emformer_transducer_stateless2/export-for-ncnn.py`.
   - `2=32`, 2 is the key and 32 is the value of the parameter `--memory-size` that you provided when running `conv_emformer_transducer_stateless2/export-for-ncnn.py`.
   - `3=31`, 3 is the key and 31 is the value of the parameter `--cnn-module-kernel` that you provided when running `conv_emformer_transducer_stateless2/export-for-ncnn.py`.
   - `4=8`, 4 is the key and 8 is the value of the parameter `--left-context-length` that you provided when running `conv_emformer_transducer_stateless2/export-for-ncnn.py`.
   - `5=32`, 5 is the key and 32 is the value of the parameter `--chunk-length` that you provided when running `conv_emformer_transducer_stateless2/export-for-ncnn.py`.
   - `6=8`, 6 is the key and 8 is the value of the parameter `--right-context-length` that you provided when running `conv_emformer_transducer_stateless2/export-for-ncnn.py`.
   - `7=512`, 7 is the key and 512 is the value of the parameter `--encoder-dim` that you provided when running `conv_emformer_transducer_stateless2/export-for-ncnn.py`.

   For ease of reference, we list the key-value pairs that you need to add in the following table. If your model has a different setting, please change the values for `SherpaMetaData` accordingly. Otherwise, you will be `SAD`.

key	value
0	1 (fixed)
1	`--num-encoder-layers`
2	`--memory-size`
3	`--cnn-module-kernel`
4	`--left-context-length`
5	`--chunk-length`
6	`--right-context-length`
7	`--encoder-dim`

4. `Input in0 0 1 in0`. No need to change it.

Caution: When you add a new layer `SherpaMetaData`, please remember to update the number of layers. In our case, update `1060` to `1061`. Otherwise, you will be SAD later.

Hint: After adding the new layer `SherpaMetaData`, you cannot use this model with `streaming-ncnn-decode.py` anymore since `SherpaMetaData` is supported only in sherpa-ncnn.

Hint: ncnn is very flexible. You can add new layers to it just by text-editing the `param` file! You don't need to change the `bin` file.

Now you can use this model in sherpa-ncnn. Please refer to the following documentation:

- Linux/macOS/Windows/arm/aarch64: https://k2-fsa.github.io/sherpa/ncnn/install/index.html
- `Android`: https://k2-fsa.github.io/sherpa/ncnn/android/index.html
- `iOS`: https://k2-fsa.github.io/sherpa/ncnn/ios/index.html
- Python: https://k2-fsa.github.io/sherpa/ncnn/python/index.html

We have a list of pre-trained models that have been exported for sherpa-ncnn:

- https://k2-fsa.github.io/sherpa/ncnn/pretrained_models/index.html

  You can find more usages there.

7. (Optional) int8 quantization with sherpa-ncnn

This step is optional.

In this step, we describe how to quantize our model with `int8`.

Change *4. Export torchscript model via pnnx* to disable `fp16` when using `pnnx`:

```
cd icefall-asr-librispeech-conv-emformer-transducer-stateless2-2022-07-05/exp/

pnnx ./encoder_jit_trace-pnnx.pt fp16=0
pnnx ./decoder_jit_trace-pnnx.pt
pnnx ./joiner_jit_trace-pnnx.pt fp16=0
```

Note: We add `fp16=0` when exporting the encoder and joiner. ncnn does not support quantizing the decoder model yet. We will update this documentation once ncnn supports it. (Maybe in this year, 2023).

It will generate the following files

```
ls -lh icefall-asr-librispeech-conv-emformer-transducer-stateless2-2022-07-05/exp/*_jit_
↪trace-pnnx.ncnn.{param,bin}

-rw-r--r-- 1 kuangfangjun root 503K Jan 11 15:56 icefall-asr-librispeech-conv-emformer-
↪transducer-stateless2-2022-07-05/exp/decoder_jit_trace-pnnx.ncnn.bin
-rw-r--r-- 1 kuangfangjun root  437 Jan 11 15:56 icefall-asr-librispeech-conv-emformer-
↪transducer-stateless2-2022-07-05/exp/decoder_jit_trace-pnnx.ncnn.param
-rw-r--r-- 1 kuangfangjun root 283M Jan 11 15:56 icefall-asr-librispeech-conv-emformer-
↪transducer-stateless2-2022-07-05/exp/encoder_jit_trace-pnnx.ncnn.bin
-rw-r--r-- 1 kuangfangjun root  79K Jan 11 15:56 icefall-asr-librispeech-conv-emformer-
↪transducer-stateless2-2022-07-05/exp/encoder_jit_trace-pnnx.ncnn.param
-rw-r--r-- 1 kuangfangjun root 3.0M Jan 11 15:56 icefall-asr-librispeech-conv-emformer-
```

(continues on next page)

(continued from previous page)

```
↪transducer-stateless2-2022-07-05/exp/joiner_jit_trace-pnnx.ncnn.bin
-rw-r--r-- 1 kuangfangjun root  488 Jan 11 15:56 icefall-asr-librispeech-conv-emformer-
↪transducer-stateless2-2022-07-05/exp/joiner_jit_trace-pnnx.ncnn.param
```

Let us compare again the file sizes:

File name	File size
encoder_jit_trace-pnnx.pt	283 MB
decoder_jit_trace-pnnx.pt	1010 KB
joiner_jit_trace-pnnx.pt	3.0 MB
encoder_jit_trace-pnnx.ncnn.bin (fp16)	142 MB
decoder_jit_trace-pnnx.ncnn.bin (fp16)	503 KB
joiner_jit_trace-pnnx.ncnn.bin (fp16)	1.5 MB
encoder_jit_trace-pnnx.ncnn.bin (fp32)	283 MB
joiner_jit_trace-pnnx.ncnn.bin (fp32)	3.0 MB

You can see that the file sizes are doubled when we disable `fp16`.

Note: You can again use `streaming-ncnn-decode.py` to test the exported models.

Next, follow *6. Modify the exported encoder for sherpa-ncnn* to modify `encoder_jit_trace-pnnx.ncnn.param`.

Change

```
7767517
1060 1342
Input                    in0                      0 1 in0
```

to

```
7767517
1061 1342
SherpaMetaData           sherpa_meta_data1        0 0 0=1 1=12 2=32 3=31 4=8 5=32 6=8␣
↪7=512
Input                    in0                      0 1 in0
```

Caution: Please follow *6. Modify the exported encoder for sherpa-ncnn* to change the values for `SherpaMetaData` if your model uses a different setting.

Next, let us compile sherpa-ncnn since we will quantize our models within sherpa-ncnn.

```
# We will download sherpa-ncnn to $HOME/open-source/
# You can change it to anywhere you like.
cd $HOME
mkdir -p open-source

cd open-source
git clone https://github.com/k2-fsa/sherpa-ncnn
cd sherpa-ncnn
```

(continues on next page)

(continued from previous page)

```
mkdir build
cd build
cmake ..
make -j 4

./bin/generate-int8-scale-table

export PATH=$HOME/open-source/sherpa-ncnn/build/bin:$PATH
```

The output of the above commands are:

```
(py38) kuangfangjun:build$ generate-int8-scale-table
Please provide 10 arg. Currently given: 1
Usage:
generate-int8-scale-table encoder.param encoder.bin decoder.param decoder.bin joiner.
↪param joiner.bin encoder-scale-table.txt joiner-scale-table.txt wave_filenames.txt

Each line in wave_filenames.txt is a path to some 16k Hz mono wave file.
```

We need to create a file `wave_filenames.txt`, in which we need to put some calibration wave files. For testing purpose, we put the `test_wavs` from the pre-trained model repository https://huggingface.co/Zengwei/icefall-asr-librispeech-conv-emformer-transducer-stateless2-2022-07-05

```
cd egs/librispeech/ASR
cd icefall-asr-librispeech-conv-emformer-transducer-stateless2-2022-07-05/exp/

cat <<EOF > wave_filenames.txt
../test_wavs/1089-134686-0001.wav
../test_wavs/1221-135766-0001.wav
../test_wavs/1221-135766-0002.wav
EOF
```

Now we can calculate the scales needed for quantization with the calibration data:

```
cd egs/librispeech/ASR
cd icefall-asr-librispeech-conv-emformer-transducer-stateless2-2022-07-05/exp/

generate-int8-scale-table \
  ./encoder_jit_trace-pnnx.ncnn.param \
  ./encoder_jit_trace-pnnx.ncnn.bin \
  ./decoder_jit_trace-pnnx.ncnn.param \
  ./decoder_jit_trace-pnnx.ncnn.bin \
  ./joiner_jit_trace-pnnx.ncnn.param \
  ./joiner_jit_trace-pnnx.ncnn.bin \
  ./encoder-scale-table.txt \
  ./joiner-scale-table.txt \
  ./wave_filenames.txt
```

The output logs are in the following:

```
Don't Use GPU. has_gpu: 0, config.use_vulkan_compute: 1
num encoder conv layers: 88
num joiner conv layers: 3
```

(continues on next page)

(continued from previous page)

```
num files: 3
Processing ../test_wavs/1089-134686-0001.wav
Processing ../test_wavs/1221-135766-0001.wav
Processing ../test_wavs/1221-135766-0002.wav
Processing ../test_wavs/1089-134686-0001.wav
Processing ../test_wavs/1221-135766-0001.wav
Processing ../test_wavs/1221-135766-0002.wav
----------encoder----------
conv_87                                  : max = 15.942385        threshold = 15.938493 ␣
↪        scale = 7.968131
conv_88                                  : max = 35.442448        threshold = 15.549335 ␣
↪        scale = 8.167552
conv_89                                  : max = 23.228289        threshold = 8.001738  ␣
↪        scale = 15.871552
linear_90                                : max = 3.976146         threshold = 1.101789  ␣
↪        scale = 115.267128
linear_91                                : max = 6.962030         threshold = 5.162033  ␣
↪        scale = 24.602713
linear_92                                : max = 12.323041        threshold = 3.853959  ␣
↪        scale = 32.953129
linear_94                                : max = 6.905416         threshold = 4.648006  ␣
↪        scale = 27.323545
linear_93                                : max = 6.905416         threshold = 5.474093  ␣
↪        scale = 23.200188
linear_95                                : max = 1.888012         threshold = 1.403563  ␣
↪        scale = 90.483986
linear_96                                : max = 6.856741         threshold = 5.398679  ␣
↪        scale = 23.524273
linear_97                                : max = 9.635942         threshold = 2.613655  ␣
↪        scale = 48.590950
linear_98                                : max = 6.460340         threshold = 5.670146  ␣
↪        scale = 22.398010
linear_99                                : max = 9.532276         threshold = 2.585537  ␣
↪        scale = 49.119396
linear_101                               : max = 6.585871         threshold = 5.719224  ␣
↪        scale = 22.205809
linear_100                               : max = 6.585871         threshold = 5.751382  ␣
↪        scale = 22.081648
linear_102                               : max = 1.593344         threshold = 1.450581  ␣
↪        scale = 87.551147
linear_103                               : max = 6.592681         threshold = 5.705824  ␣
↪        scale = 22.257959
linear_104                               : max = 8.752957         threshold = 1.980955  ␣
↪        scale = 64.110489
linear_105                               : max = 6.696240         threshold = 5.877193  ␣
↪        scale = 21.608953
linear_106                               : max = 9.059659         threshold = 2.643138  ␣
↪        scale = 48.048950
linear_108                               : max = 6.975461         threshold = 4.589567  ␣
↪        scale = 27.671457
linear_107                               : max = 6.975461         threshold = 6.190381  ␣
↪        scale = 20.515701
```

(continues on next page)

(continued from previous page)

```
linear_109                                  : max = 3.710759        threshold = 2.305635  ␣
↪       scale = 55.082436
linear_110                                  : max = 7.531228        threshold = 5.731162  ␣
↪       scale = 22.159557
linear_111                                  : max = 10.528083       threshold = 2.259322  ␣
↪       scale = 56.211544
linear_112                                  : max = 8.148807        threshold = 5.500842  ␣
↪       scale = 23.087374
linear_113                                  : max = 8.592566        threshold = 1.948851  ␣
↪       scale = 65.166611
linear_115                                  : max = 8.437109        threshold = 5.608947  ␣
↪       scale = 22.642395
linear_114                                  : max = 8.437109        threshold = 6.193942  ␣
↪       scale = 20.503904
linear_116                                  : max = 3.966980        threshold = 3.200896  ␣
↪       scale = 39.676392
linear_117                                  : max = 9.451303        threshold = 6.061664  ␣
↪       scale = 20.951344
linear_118                                  : max = 12.077262       threshold = 3.965800  ␣
↪       scale = 32.023804
linear_119                                  : max = 9.671615        threshold = 4.847613  ␣
↪       scale = 26.198460
linear_120                                  : max = 8.625638        threshold = 3.131427  ␣
↪       scale = 40.556595
linear_122                                  : max = 10.274080       threshold = 4.888716  ␣
↪       scale = 25.978189
linear_121                                  : max = 10.274080       threshold = 5.420480  ␣
↪       scale = 23.429659
linear_123                                  : max = 4.826197        threshold = 3.599617  ␣
↪       scale = 35.281532
linear_124                                  : max = 11.396383       threshold = 7.325849  ␣
↪       scale = 17.335875
linear_125                                  : max = 9.337198        threshold = 3.941410  ␣
↪       scale = 32.221970
linear_126                                  : max = 9.699965        threshold = 4.842878  ␣
↪       scale = 26.224073
linear_127                                  : max = 8.775370        threshold = 3.884215  ␣
↪       scale = 32.696438
linear_129                                  : max = 9.872276        threshold = 4.837319  ␣
↪       scale = 26.254213
linear_128                                  : max = 9.872276        threshold = 7.180057  ␣
↪       scale = 17.687883
linear_130                                  : max = 4.150427        threshold = 3.454298  ␣
↪       scale = 36.765789
linear_131                                  : max = 11.112692       threshold = 7.924847  ␣
↪       scale = 16.025545
linear_132                                  : max = 11.852893       threshold = 3.116593  ␣
↪       scale = 40.749626
linear_133                                  : max = 11.517084       threshold = 5.024665  ␣
↪       scale = 25.275314
linear_134                                  : max = 10.683807       threshold = 3.878618  ␣
↪       scale = 32.743618
```

(continues on next page)

(continued from previous page)

```
linear_136                               : max = 12.421055        threshold = 6.322729  ␣
↪       scale = 20.086264
linear_135                               : max = 12.421055        threshold = 5.309880  ␣
↪       scale = 23.917679
linear_137                               : max = 4.827781         threshold = 3.744595  ␣
↪       scale = 33.915554
linear_138                               : max = 14.422395        threshold = 7.742882  ␣
↪       scale = 16.402161
linear_139                               : max = 8.527538         threshold = 3.866123  ␣
↪       scale = 32.849449
linear_140                               : max = 12.128619        threshold = 4.657793  ␣
↪       scale = 27.266134
linear_141                               : max = 9.839593         threshold = 3.845993  ␣
↪       scale = 33.021378
linear_143                               : max = 12.442304        threshold = 7.099039  ␣
↪       scale = 17.889746
linear_142                               : max = 12.442304        threshold = 5.325038  ␣
↪       scale = 23.849592
linear_144                               : max = 5.929444         threshold = 5.618206  ␣
↪       scale = 22.605080
linear_145                               : max = 13.382126        threshold = 9.321095  ␣
↪       scale = 13.625010
linear_146                               : max = 9.894987         threshold = 3.867645  ␣
↪       scale = 32.836517
linear_147                               : max = 10.915313        threshold = 4.906028  ␣
↪       scale = 25.886522
linear_148                               : max = 9.614287         threshold = 3.908151  ␣
↪       scale = 32.496181
linear_150                               : max = 11.724932        threshold = 4.485588  ␣
↪       scale = 28.312899
linear_149                               : max = 11.724932        threshold = 5.161146  ␣
↪       scale = 24.606939
linear_151                               : max = 7.164453         threshold = 5.847355  ␣
↪       scale = 21.719223
linear_152                               : max = 13.086471        threshold = 5.984121  ␣
↪       scale = 21.222834
linear_153                               : max = 11.099524        threshold = 3.991601  ␣
↪       scale = 31.816805
linear_154                               : max = 10.054585        threshold = 4.489706  ␣
↪       scale = 28.286930
linear_155                               : max = 12.389185        threshold = 3.100321  ␣
↪       scale = 40.963501
linear_157                               : max = 9.982999         threshold = 5.154796  ␣
↪       scale = 24.637253
linear_156                               : max = 9.982999         threshold = 8.537706  ␣
↪       scale = 14.875190
linear_158                               : max = 8.420287         threshold = 6.502287  ␣
↪       scale = 19.531588
linear_159                               : max = 25.014746        threshold = 9.423280  ␣
↪       scale = 13.477261
linear_160                               : max = 45.633553        threshold = 5.715335  ␣
↪       scale = 22.220921
```

(continues on next page)

(continued from previous page)

```
linear_161                                : max = 20.371849        threshold = 5.117830  ␣
↪       scale = 24.815203
linear_162                                : max = 12.492933        threshold = 3.126283  ␣
↪       scale = 40.623318
linear_164                                : max = 20.697504        threshold = 4.825712  ␣
↪       scale = 26.317358
linear_163                                : max = 20.697504        threshold = 5.078367  ␣
↪       scale = 25.008038
linear_165                                : max = 9.023975         threshold = 6.836278  ␣
↪       scale = 18.577358
linear_166                                : max = 34.860619        threshold = 7.259792  ␣
↪       scale = 17.493614
linear_167                                : max = 30.380934        threshold = 5.496160  ␣
↪       scale = 23.107042
linear_168                                : max = 20.691216        threshold = 4.733317  ␣
↪       scale = 26.831076
linear_169                                : max = 9.723948         threshold = 3.952728  ␣
↪       scale = 32.129707
linear_171                                : max = 21.034811        threshold = 5.366547  ␣
↪       scale = 23.665123
linear_170                                : max = 21.034811        threshold = 5.356277  ␣
↪       scale = 23.710501
linear_172                                : max = 10.556884        threshold = 5.729481  ␣
↪       scale = 22.166058
linear_173                                : max = 20.033039        threshold = 10.207264 ␣
↪       scale = 12.442120
linear_174                                : max = 11.597379        threshold = 2.658676  ␣
↪       scale = 47.768131
----------joiner----------
linear_2                                  : max = 19.293503        threshold = 14.305265 ␣
↪       scale = 8.877850
linear_1                                  : max = 10.812222        threshold = 8.766452  ␣
↪       scale = 14.487047
linear_3                                  : max = 0.999999         threshold = 0.999755  ␣
↪       scale = 127.031174
ncnn int8 calibration table create success, best wish for your int8 inference has a low␣
↪accuracy loss...\(^0^)/...233...
```

It generates the following two files:

```
$ ls -lh encoder-scale-table.txt joiner-scale-table.txt
-rw-r--r-- 1 kuangfangjun root 955K Jan 11 17:28 encoder-scale-table.txt
-rw-r--r-- 1 kuangfangjun root  18K Jan 11 17:28 joiner-scale-table.txt
```

Caution: Definitely, you need more calibration data to compute the scale table.

Finally, let us use the scale table to quantize our models into `int8`.

```
ncnn2int8

usage: ncnn2int8 [inparam] [inbin] [outparam] [outbin] [calibration table]
```

First, we quantize the encoder model:

```
cd egs/librispeech/ASR
cd icefall-asr-librispeech-conv-emformer-transducer-stateless2-2022-07-05/exp/

ncnn2int8 \
  ./encoder_jit_trace-pnnx.ncnn.param \
  ./encoder_jit_trace-pnnx.ncnn.bin \
  ./encoder_jit_trace-pnnx.ncnn.int8.param \
  ./encoder_jit_trace-pnnx.ncnn.int8.bin \
  ./encoder-scale-table.txt
```

Next, we quantize the joiner model:

```
ncnn2int8 \
  ./joiner_jit_trace-pnnx.ncnn.param \
  ./joiner_jit_trace-pnnx.ncnn.bin \
  ./joiner_jit_trace-pnnx.ncnn.int8.param \
  ./joiner_jit_trace-pnnx.ncnn.int8.bin \
  ./joiner-scale-table.txt
```

The above two commands generate the following 4 files:

```
-rw-r--r-- 1 kuangfangjun root  99M Jan 11 17:34 encoder_jit_trace-pnnx.ncnn.int8.bin
-rw-r--r-- 1 kuangfangjun root  78K Jan 11 17:34 encoder_jit_trace-pnnx.ncnn.int8.param
-rw-r--r-- 1 kuangfangjun root 774K Jan 11 17:35 joiner_jit_trace-pnnx.ncnn.int8.bin
-rw-r--r-- 1 kuangfangjun root  496 Jan 11 17:35 joiner_jit_trace-pnnx.ncnn.int8.param
```

Congratulations! You have successfully quantized your model from `float32` to `int8`.

Caution: `ncnn.int8.param` and `ncnn.int8.bin` must be used in pairs.

You can replace `ncnn.param` and `ncnn.bin` with `ncnn.int8.param` and `ncnn.int8.bin` in sherpa-ncnn if you like.

For instance, to use only the `int8` encoder in `sherpa-ncnn`, you can replace the following invocation:

```
cd egs/librispeech/ASR
cd icefall-asr-librispeech-conv-emformer-transducer-stateless2-2022-07-05/
→exp/

sherpa-ncnn \
  ../data/lang_bpe_500/tokens.txt \
  ./encoder_jit_trace-pnnx.ncnn.param \
  ./encoder_jit_trace-pnnx.ncnn.bin \
  ./decoder_jit_trace-pnnx.ncnn.param \
  ./decoder_jit_trace-pnnx.ncnn.bin \
  ./joiner_jit_trace-pnnx.ncnn.param \
  ./joiner_jit_trace-pnnx.ncnn.bin \
  ../test_wavs/1089-134686-0001.wav
```

with

```
cd egs/librispeech/ASR
cd icefall-asr-librispeech-conv-emformer-transducer-stateless2-2022-07-05/
↪exp/

sherpa-ncnn \
  ../data/lang_bpe_500/tokens.txt \
  ./encoder_jit_trace-pnnx.ncnn.int8.param \
  ./encoder_jit_trace-pnnx.ncnn.int8.bin \
  ./decoder_jit_trace-pnnx.ncnn.param \
  ./decoder_jit_trace-pnnx.ncnn.bin \
  ./joiner_jit_trace-pnnx.ncnn.param \
  ./joiner_jit_trace-pnnx.ncnn.bin \
  ../test_wavs/1089-134686-0001.wav
```

The following table compares again the file sizes:

File name	File size
encoder_jit_trace-pnnx.pt	283 MB
decoder_jit_trace-pnnx.pt	1010 KB
joiner_jit_trace-pnnx.pt	3.0 MB
encoder_jit_trace-pnnx.ncnn.bin (fp16)	142 MB
decoder_jit_trace-pnnx.ncnn.bin (fp16)	503 KB
joiner_jit_trace-pnnx.ncnn.bin (fp16)	1.5 MB
encoder_jit_trace-pnnx.ncnn.bin (fp32)	283 MB
joiner_jit_trace-pnnx.ncnn.bin (fp32)	3.0 MB
encoder_jit_trace-pnnx.ncnn.int8.bin	99 MB
joiner_jit_trace-pnnx.ncnn.int8.bin	774 KB

You can see that the file sizes of the model after `int8` quantization are much smaller.

Hint: Currently, only linear layers and convolutional layers are quantized with `int8`, so you don't see an exact `4x` reduction in file sizes.

Note: You need to test the recognition accuracy after `int8` quantization.

You can find the speed comparison at https://github.com/k2-fsa/sherpa-ncnn/issues/44.

That's it! Have fun with sherpa-ncnn!

5.5.3 Export LSTM transducer models to ncnn

We use the pre-trained model from the following repository as an example:

https://huggingface.co/csukuangfj/icefall-asr-librispeech-lstm-transducer-stateless2-2022-09-03

We will show you step by step how to export it to ncnn and run it with sherpa-ncnn.

Hint: We use `Ubuntu 18.04`, `torch 1.13`, and `Python 3.8` for testing.

Caution: Please use a more recent version of PyTorch. For instance, `torch 1.8` may `not` work.

1. Download the pre-trained model

Hint: You have to install git-lfs before you continue.

```
cd egs/librispeech/ASR
GIT_LFS_SKIP_SMUDGE=1 git clone https://huggingface.co/csukuangfj/icefall-asr-
↪librispeech-lstm-transducer-stateless2-2022-09-03
cd icefall-asr-librispeech-lstm-transducer-stateless2-2022-09-03

git lfs pull --include "exp/pretrained-iter-468000-avg-16.pt"
git lfs pull --include "data/lang_bpe_500/bpe.model"

cd ..
```

Note: We downloaded `exp/pretrained-xxx.pt`, not `exp/cpu-jit_xxx.pt`.

In the above code, we downloaded the pre-trained model into the directory `egs/librispeech/ASR/icefall-asr-librispeech-lstm-transducer-stateless2-2022-09-03`.

2. Install ncnn and pnnx

Please refer to *2. Install ncnn and pnnx* .

3. Export the model via torch.jit.trace()

First, let us rename our pre-trained model:

```
cd egs/librispeech/ASR

cd icefall-asr-librispeech-lstm-transducer-stateless2-2022-09-03/exp

ln -s pretrained-iter-468000-avg-16.pt epoch-99.pt

cd ../..
```

Next, we use the following code to export our model:

```
dir=./icefall-asr-librispeech-lstm-transducer-stateless2-2022-09-03

./lstm_transducer_stateless2/export-for-ncnn.py \
  --exp-dir $dir/exp \
  --tokens $dir/data/lang_bpe_500/tokens.txt \
  --epoch 99 \
  --avg 1 \
  --use-averaged-model 0 \
  --num-encoder-layers 12 \
  --encoder-dim 512 \
  --rnn-hidden-size 1024
```

Hint: We have renamed our model to `epoch-99.pt` so that we can use `--epoch 99`. There is only one pre-trained model, so we use `--avg 1 --use-averaged-model 0`.

If you have trained a model by yourself and if you have all checkpoints available, please first use `decode.py` to tune `--epoch --avg` and select the best combination with with `--use-averaged-model 1`.

Note: You will see the following log output:

```
2023-02-17 11:22:42,862 INFO [export-for-ncnn.py:222] device: cpu
2023-02-17 11:22:42,865 INFO [export-for-ncnn.py:231] {'best_train_loss': inf, 'best_
↪valid_loss': inf, 'best_train_epoch': -1, 'best_valid_epoch': -1, 'batch_idx_train': 0,
↪ 'log_interval': 50, 'reset_interval': 200, 'valid_interval': 3000, 'feature_dim': 80,
↪'subsampling_factor': 4, 'dim_feedforward': 2048, 'decoder_dim': 512, 'joiner_dim':␣
↪512, 'is_pnnx': False, 'model_warm_step': 3000, 'env_info': {'k2-version': '1.23.4',
↪'k2-build-type': 'Release', 'k2-with-cuda': True, 'k2-git-sha1':
↪'62e404dd3f3a811d73e424199b3408e309c06e1a', 'k2-git-date': 'Mon Jan 30 10:26:16 2023',
↪'lhotse-version': '1.12.0.dev+missing.version.file', 'torch-version': '1.10.0+cu102',
↪'torch-cuda-available': False, 'torch-cuda-version': '10.2', 'python-version': '3.8',
↪'icefall-git-branch': 'master', 'icefall-git-sha1': '6d7a559-dirty', 'icefall-git-date
↪': 'Thu Feb 16 19:47:54 2023', 'icefall-path': '/star-fj/fangjun/open-source/icefall-2
↪', 'k2-path': '/star-fj/fangjun/open-source/k2/k2/python/k2/__init__.py', 'lhotse-path
↪': '/star-fj/fangjun/open-source/lhotse/lhotse/__init__.py', 'hostname': 'de-74279-k2-
↪train-3-1220120619-7695ff496b-s9n4w', 'IP address': '10.177.6.147'}, 'epoch': 99, 'iter
↪': 0, 'avg': 1, 'exp_dir': PosixPath('icefall-asr-librispeech-lstm-transducer-
↪stateless2-2022-09-03/exp'), 'bpe_model': './icefall-asr-librispeech-lstm-transducer-
↪stateless2-2022-09-03/data/lang_bpe_500/bpe.model', 'context_size': 2, 'use_averaged_
↪model': False, 'num_encoder_layers': 12, 'encoder_dim': 512, 'rnn_hidden_size': 1024,
↪'aux_layer_period': 0, 'blank_id': 0, 'vocab_size': 500}
2023-02-17 11:22:42,865 INFO [export-for-ncnn.py:235] About to create model
2023-02-17 11:22:43,239 INFO [train.py:472] Disable giga
2023-02-17 11:22:43,249 INFO [checkpoint.py:112] Loading checkpoint from icefall-asr-
↪librispeech-lstm-transducer-stateless2-2022-09-03/exp/epoch-99.pt
2023-02-17 11:22:44,595 INFO [export-for-ncnn.py:324] encoder parameters: 83137520
2023-02-17 11:22:44,596 INFO [export-for-ncnn.py:325] decoder parameters: 257024
2023-02-17 11:22:44,596 INFO [export-for-ncnn.py:326] joiner parameters: 781812
2023-02-17 11:22:44,596 INFO [export-for-ncnn.py:327] total parameters: 84176356
2023-02-17 11:22:44,596 INFO [export-for-ncnn.py:329] Using torch.jit.trace()
2023-02-17 11:22:44,596 INFO [export-for-ncnn.py:331] Exporting encoder
```

(continues on next page)

(continued from previous page)

```
2023-02-17 11:22:48,182 INFO [export-for-ncnn.py:158] Saved to icefall-asr-librispeech-
↪lstm-transducer-stateless2-2022-09-03/exp/encoder_jit_trace-pnnx.pt
2023-02-17 11:22:48,183 INFO [export-for-ncnn.py:335] Exporting decoder
/star-fj/fangjun/open-source/icefall-2/egs/librispeech/ASR/lstm_transducer_stateless2/
↪decoder.py:101: TracerWarning: Converting a tensor to a Python boolean might cause the␣
↪trace to be incorrect. We can't record the data flow of Python values, so this value␣
↪will be treated as a constant in the future. This means that the trace might not␣
↪generalize to other inputs!
  need_pad = bool(need_pad)
2023-02-17 11:22:48,259 INFO [export-for-ncnn.py:180] Saved to icefall-asr-librispeech-
↪lstm-transducer-stateless2-2022-09-03/exp/decoder_jit_trace-pnnx.pt
2023-02-17 11:22:48,259 INFO [export-for-ncnn.py:339] Exporting joiner
2023-02-17 11:22:48,304 INFO [export-for-ncnn.py:207] Saved to icefall-asr-librispeech-
↪lstm-transducer-stateless2-2022-09-03/exp/joiner_jit_trace-pnnx.pt
```

The log shows the model has `84176356` parameters, i.e., `~84 M`.

```
ls -lh icefall-asr-librispeech-lstm-transducer-stateless2-2022-09-03/exp/pretrained-iter-
↪468000-avg-16.pt

-rw-r--r-- 1 kuangfangjun root 324M Feb 17 10:34 icefall-asr-librispeech-lstm-transducer-
↪stateless2-2022-09-03/exp/pretrained-iter-468000-avg-16.pt
```

You can see that the file size of the pre-trained model is `324 MB`, which is roughly equal to `84176356*4/1024/1024 = 321.107 MB`.

After running `lstm_transducer_stateless2/export-for-ncnn.py`, we will get the following files:

```
ls -lh icefall-asr-librispeech-lstm-transducer-stateless2-2022-09-03/exp/*pnnx.pt

-rw-r--r-- 1 kuangfangjun root 1010K Feb 17 11:22 icefall-asr-librispeech-lstm-
↪transducer-stateless2-2022-09-03/exp/decoder_jit_trace-pnnx.pt
-rw-r--r-- 1 kuangfangjun root  318M Feb 17 11:22 icefall-asr-librispeech-lstm-
↪transducer-stateless2-2022-09-03/exp/encoder_jit_trace-pnnx.pt
-rw-r--r-- 1 kuangfangjun root  3.0M Feb 17 11:22 icefall-asr-librispeech-lstm-
↪transducer-stateless2-2022-09-03/exp/joiner_jit_trace-pnnx.pt
```

4. Export torchscript model via pnnx

Hint: Make sure you have set up the `PATH` environment variable in *2. Install ncnn and pnnx*. Otherwise, it will throw an error saying that `pnnx` could not be found.

Now, it's time to export our models to ncnn via `pnnx`.

```
cd icefall-asr-librispeech-lstm-transducer-stateless2-2022-09-03/exp/

pnnx ./encoder_jit_trace-pnnx.pt
pnnx ./decoder_jit_trace-pnnx.pt
pnnx ./joiner_jit_trace-pnnx.pt
```

It will generate the following files:

```
ls -lh  icefall-asr-librispeech-lstm-transducer-stateless2-2022-09-03/exp/*ncnn*{bin,
↪param}

-rw-r--r-- 1 kuangfangjun root 503K Feb 17 11:32 icefall-asr-librispeech-lstm-transducer-
↪stateless2-2022-09-03/exp/decoder_jit_trace-pnnx.ncnn.bin
-rw-r--r-- 1 kuangfangjun root  437 Feb 17 11:32 icefall-asr-librispeech-lstm-transducer-
↪stateless2-2022-09-03/exp/decoder_jit_trace-pnnx.ncnn.param
-rw-r--r-- 1 kuangfangjun root 159M Feb 17 11:32 icefall-asr-librispeech-lstm-transducer-
↪stateless2-2022-09-03/exp/encoder_jit_trace-pnnx.ncnn.bin
-rw-r--r-- 1 kuangfangjun root  21K Feb 17 11:32 icefall-asr-librispeech-lstm-transducer-
↪stateless2-2022-09-03/exp/encoder_jit_trace-pnnx.ncnn.param
-rw-r--r-- 1 kuangfangjun root 1.5M Feb 17 11:33 icefall-asr-librispeech-lstm-transducer-
↪stateless2-2022-09-03/exp/joiner_jit_trace-pnnx.ncnn.bin
-rw-r--r-- 1 kuangfangjun root  488 Feb 17 11:33 icefall-asr-librispeech-lstm-transducer-
↪stateless2-2022-09-03/exp/joiner_jit_trace-pnnx.ncnn.param
```

There are two types of files:

- `param`: It is a text file containing the model architectures. You can use a text editor to view its content.
- `bin`: It is a binary file containing the model parameters.

We compare the file sizes of the models below before and after converting via `pnnx`:

File name	File size
encoder_jit_trace-pnnx.pt	318 MB
decoder_jit_trace-pnnx.pt	1010 KB
joiner_jit_trace-pnnx.pt	3.0 MB
encoder_jit_trace-pnnx.ncnn.bin	159 MB
decoder_jit_trace-pnnx.ncnn.bin	503 KB
joiner_jit_trace-pnnx.ncnn.bin	1.5 MB

You can see that the file sizes of the models after conversion are about one half of the models before conversion:

- encoder: 318 MB vs 159 MB
- decoder: 1010 KB vs 503 KB
- joiner: 3.0 MB vs 1.5 MB

The reason is that by default `pnnx` converts `float32` parameters to `float16`. A `float32` parameter occupies 4 bytes, while it is 2 bytes for `float16`. Thus, it is `twice smaller` after conversion.

Hint: If you use `pnnx ./encoder_jit_trace-pnnx.pt fp16=0`, then `pnnx` won't convert `float32` to `float16`.

5. Test the exported models in icefall

Note: We assume you have set up the environment variable PYTHONPATH when building ncnn.

Now we have successfully converted our pre-trained model to ncnn format. The generated 6 files are what we need. You can use the following code to test the converted models:

```
python3 ./lstm_transducer_stateless2/streaming-ncnn-decode.py \
  --tokens ./icefall-asr-librispeech-lstm-transducer-stateless2-2022-09-03/data/lang_bpe_
↪500/tokens.txt \
  --encoder-param-filename ./icefall-asr-librispeech-lstm-transducer-stateless2-2022-09-
↪03/exp/encoder_jit_trace-pnnx.ncnn.param \
  --encoder-bin-filename ./icefall-asr-librispeech-lstm-transducer-stateless2-2022-09-03/
↪exp/encoder_jit_trace-pnnx.ncnn.bin \
  --decoder-param-filename ./icefall-asr-librispeech-lstm-transducer-stateless2-2022-09-
↪03/exp/decoder_jit_trace-pnnx.ncnn.param \
  --decoder-bin-filename ./icefall-asr-librispeech-lstm-transducer-stateless2-2022-09-03/
↪exp/decoder_jit_trace-pnnx.ncnn.bin \
  --joiner-param-filename ./icefall-asr-librispeech-lstm-transducer-stateless2-2022-09-
↪03/exp/joiner_jit_trace-pnnx.ncnn.param \
  --joiner-bin-filename ./icefall-asr-librispeech-lstm-transducer-stateless2-2022-09-03/
↪exp/joiner_jit_trace-pnnx.ncnn.bin \
  ./icefall-asr-librispeech-lstm-transducer-stateless2-2022-09-03/test_wavs/1089-134686-
↪0001.wav
```

Hint: ncnn supports only `batch size == 1`, so `streaming-ncnn-decode.py` accepts only 1 wave file as input.

The output is given below:

```
2023-02-17 11:37:30,861 INFO [streaming-ncnn-decode.py:255] {'tokens': './icefall-asr-
↪librispeech-lstm-transducer-stateless2-2022-09-03/data/lang_bpe_500/tokens.txt',
↪'encoder_param_filename': './icefall-asr-librispeech-lstm-transducer-stateless2-2022-
↪09-03/exp/encoder_jit_trace-pnnx.ncnn.param', 'encoder_bin_filename': './icefall-asr-
↪librispeech-lstm-transducer-stateless2-2022-09-03/exp/encoder_jit_trace-pnnx.ncnn.bin',
↪ 'decoder_param_filename': './icefall-asr-librispeech-lstm-transducer-stateless2-2022-
↪09-03/exp/decoder_jit_trace-pnnx.ncnn.param', 'decoder_bin_filename': './icefall-asr-
↪librispeech-lstm-transducer-stateless2-2022-09-03/exp/decoder_jit_trace-pnnx.ncnn.bin',
↪ 'joiner_param_filename': './icefall-asr-librispeech-lstm-transducer-stateless2-2022-
↪09-03/exp/joiner_jit_trace-pnnx.ncnn.param', 'joiner_bin_filename': './icefall-asr-
↪librispeech-lstm-transducer-stateless2-2022-09-03/exp/joiner_jit_trace-pnnx.ncnn.bin',
↪'sound_filename': './icefall-asr-librispeech-lstm-transducer-stateless2-2022-09-03/
↪test_wavs/1089-134686-0001.wav'}
2023-02-17 11:37:31,425 INFO [streaming-ncnn-decode.py:263] Constructing Fbank computer
2023-02-17 11:37:31,427 INFO [streaming-ncnn-decode.py:266] Reading sound files: ./
↪icefall-asr-librispeech-lstm-transducer-stateless2-2022-09-03/test_wavs/1089-134686-
↪0001.wav
2023-02-17 11:37:31,431 INFO [streaming-ncnn-decode.py:271] torch.Size([106000])
2023-02-17 11:37:34,115 INFO [streaming-ncnn-decode.py:342] ./icefall-asr-librispeech-
↪lstm-transducer-stateless2-2022-09-03/test_wavs/1089-134686-0001.wav
2023-02-17 11:37:34,115 INFO [streaming-ncnn-decode.py:343] AFTER EARLY NIGHTFALL THE␣
↪YELLOW LAMPS WOULD LIGHT UP HERE AND THERE THE SQUALID QUARTER OF THE BROTHELS
```

Congratulations! You have successfully exported a model from PyTorch to ncnn!

6. Modify the exported encoder for sherpa-ncnn

In order to use the exported models in sherpa-ncnn, we have to modify `encoder_jit_trace-pnnx.ncnn.param`.

Let us have a look at the first few lines of `encoder_jit_trace-pnnx.ncnn.param`:

```
7767517
267 379
Input                    in0                      0 1 in0
```

Explanation of the above three lines:

1. `7767517`, it is a magic number and should not be changed.
2. `267 379`, the first number `267` specifies the number of layers in this file, while `379` specifies the number of intermediate outputs of this file
3. `Input in0 0 1 in0`, `Input` is the layer type of this layer; `in0` is the layer name of this layer; `0` means this layer has no input; `1` means this layer has one output; `in0` is the output name of this layer.

We need to add 1 extra line and also increment the number of layers. The result looks like below:

```
7767517
268 379
SherpaMetaData           sherpa_meta_data1        0 0 0=3 1=12 2=512 3=1024
Input                    in0                      0 1 in0
```

Explanation

1. `7767517`, it is still the same
2. `268 379`, we have added an extra layer, so we need to update `267` to `268`. We don't need to change `379` since the newly added layer has no inputs or outputs.
3. `SherpaMetaData sherpa_meta_data1 0 0 0=3 1=12 2=512 3=1024` This line is newly added. Its explanation is given below:
   - `SherpaMetaData` is the type of this layer. Must be `SherpaMetaData`.
   - `sherpa_meta_data1` is the name of this layer. Must be `sherpa_meta_data1`.
   - `0 0` means this layer has no inputs or output. Must be `0 0`
   - `0=3`, 0 is the key and 3 is the value. MUST be `0=3`
   - `1=12`, 1 is the key and 12 is the value of the parameter `--num-encoder-layers` that you provided when running `./lstm_transducer_stateless2/export-for-ncnn.py`.
   - `2=512`, 2 is the key and 512 is the value of the parameter `--encoder-dim` that you provided when running `./lstm_transducer_stateless2/export-for-ncnn.py`.
   - `3=1024`, 3 is the key and 1024 is the value of the parameter `--rnn-hidden-size` that you provided when running `./lstm_transducer_stateless2/export-for-ncnn.py`.

   For ease of reference, we list the key-value pairs that you need to add in the following table. If your model has a different setting, please change the values for `SherpaMetaData` accordingly. Otherwise, you will be `SAD`.

key	value
0	3 (fixed)
1	`--num-encoder-layers`
2	`--encoder-dim`
3	`--rnn-hidden-size`

4. `Input in0 0 1 in0`. No need to change it.

Caution: When you add a new layer `SherpaMetaData`, please remember to update the number of layers. In our case, update `267` to `268`. Otherwise, you will be SAD later.

Hint: After adding the new layer `SherpaMetaData`, you cannot use this model with `streaming-ncnn-decode.py` anymore since `SherpaMetaData` is supported only in sherpa-ncnn.

Hint: ncnn is very flexible. You can add new layers to it just by text-editing the `param` file! You don't need to change the `bin` file.

Now you can use this model in sherpa-ncnn. Please refer to the following documentation:

- Linux/macOS/Windows/arm/aarch64: https://k2-fsa.github.io/sherpa/ncnn/install/index.html
- `Android`: https://k2-fsa.github.io/sherpa/ncnn/android/index.html
- `iOS`: https://k2-fsa.github.io/sherpa/ncnn/ios/index.html
- Python: https://k2-fsa.github.io/sherpa/ncnn/python/index.html

We have a list of pre-trained models that have been exported for sherpa-ncnn:

- https://k2-fsa.github.io/sherpa/ncnn/pretrained_models/index.html

  You can find more usages there.

7. (Optional) int8 quantization with sherpa-ncnn

This step is optional.

In this step, we describe how to quantize our model with `int8`.

Change *4. Export torchscript model via pnnx* to disable `fp16` when using `pnnx`:

```
cd icefall-asr-librispeech-lstm-transducer-stateless2-2022-09-03/exp/

pnnx ./encoder_jit_trace-pnnx.pt fp16=0
pnnx ./decoder_jit_trace-pnnx.pt
pnnx ./joiner_jit_trace-pnnx.pt fp16=0
```

Note: We add `fp16=0` when exporting the encoder and joiner. ncnn does not support quantizing the decoder model yet. We will update this documentation once ncnn supports it. (Maybe in this year, 2023).

```
ls -lh icefall-asr-librispeech-lstm-transducer-stateless2-2022-09-03/exp/*_jit_trace-
↪pnnx.ncnn.{param,bin}

-rw-r--r-- 1 kuangfangjun root 503K Feb 17 11:32 icefall-asr-librispeech-lstm-transducer-
↪stateless2-2022-09-03/exp/decoder_jit_trace-pnnx.ncnn.bin
-rw-r--r-- 1 kuangfangjun root  437 Feb 17 11:32 icefall-asr-librispeech-lstm-transducer-
↪stateless2-2022-09-03/exp/decoder_jit_trace-pnnx.ncnn.param
-rw-r--r-- 1 kuangfangjun root 317M Feb 17 11:54 icefall-asr-librispeech-lstm-transducer-
↪stateless2-2022-09-03/exp/encoder_jit_trace-pnnx.ncnn.bin
-rw-r--r-- 1 kuangfangjun root  21K Feb 17 11:54 icefall-asr-librispeech-lstm-transducer-
↪stateless2-2022-09-03/exp/encoder_jit_trace-pnnx.ncnn.param
-rw-r--r-- 1 kuangfangjun root 3.0M Feb 17 11:54 icefall-asr-librispeech-lstm-transducer-
↪stateless2-2022-09-03/exp/joiner_jit_trace-pnnx.ncnn.bin
-rw-r--r-- 1 kuangfangjun root  488 Feb 17 11:54 icefall-asr-librispeech-lstm-transducer-
↪stateless2-2022-09-03/exp/joiner_jit_trace-pnnx.ncnn.param
```

Let us compare again the file sizes:

File name	File size
encoder_jit_trace-pnnx.pt	318 MB
decoder_jit_trace-pnnx.pt	1010 KB
joiner_jit_trace-pnnx.pt	3.0 MB
encoder_jit_trace-pnnx.ncnn.bin (fp16)	159 MB
decoder_jit_trace-pnnx.ncnn.bin (fp16)	503 KB
joiner_jit_trace-pnnx.ncnn.bin (fp16)	1.5 MB
encoder_jit_trace-pnnx.ncnn.bin (fp32)	317 MB
joiner_jit_trace-pnnx.ncnn.bin (fp32)	3.0 MB

You can see that the file sizes are doubled when we disable `fp16`.

Note: You can again use `streaming-ncnn-decode.py` to test the exported models.

Next, follow *6. Modify the exported encoder for sherpa-ncnn* to modify `encoder_jit_trace-pnnx.ncnn.param`.

Change

```
7767517
267 379
Input                    in0                      0 1 in0
```

to

```
7767517
268 379
SherpaMetaData           sherpa_meta_data1        0 0 0=3 1=12 2=512 3=1024
Input                    in0                      0 1 in0
```

Caution: Please follow *6. Modify the exported encoder for sherpa-ncnn* to change the values for `SherpaMetaData` if your model uses a different setting.

Next, let us compile sherpa-ncnn since we will quantize our models within sherpa-ncnn.

```
# We will download sherpa-ncnn to $HOME/open-source/
# You can change it to anywhere you like.
cd $HOME
mkdir -p open-source

cd open-source
git clone https://github.com/k2-fsa/sherpa-ncnn
cd sherpa-ncnn
mkdir build
cd build
cmake ..
make -j 4

./bin/generate-int8-scale-table

export PATH=$HOME/open-source/sherpa-ncnn/build/bin:$PATH
```

The output of the above commands are:

```
(py38) kuangfangjun:build$ generate-int8-scale-table
Please provide 10 arg. Currently given: 1
Usage:
generate-int8-scale-table encoder.param encoder.bin decoder.param decoder.bin joiner.
↪param joiner.bin encoder-scale-table.txt joiner-scale-table.txt wave_filenames.txt

Each line in wave_filenames.txt is a path to some 16k Hz mono wave file.
```

We need to create a file `wave_filenames.txt`, in which we need to put some calibration wave files. For testing purpose, we put the `test_wavs` from the pre-trained model repository https://huggingface.co/csukuangfj/icefall-asr-librispeech-lstm-transducer-stateless2-2022-09-03

```
cd egs/librispeech/ASR
cd icefall-asr-librispeech-lstm-transducer-stateless2-2022-09-03/exp/

cat <<EOF > wave_filenames.txt
../test_wavs/1089-134686-0001.wav
../test_wavs/1221-135766-0001.wav
../test_wavs/1221-135766-0002.wav
EOF
```

Now we can calculate the scales needed for quantization with the calibration data:

```
cd egs/librispeech/ASR
cd icefall-asr-librispeech-lstm-transducer-stateless2-2022-09-03/exp/

generate-int8-scale-table \
  ./encoder_jit_trace-pnnx.ncnn.param \
  ./encoder_jit_trace-pnnx.ncnn.bin \
  ./decoder_jit_trace-pnnx.ncnn.param \
  ./decoder_jit_trace-pnnx.ncnn.bin \
  ./joiner_jit_trace-pnnx.ncnn.param \
  ./joiner_jit_trace-pnnx.ncnn.bin \
  ./encoder-scale-table.txt \
```

(continues on next page)

(continued from previous page)

```
  ./joiner-scale-table.txt \
  ./wave_filenames.txt
```

The output logs are in the following:

```
Don't Use GPU. has_gpu: 0, config.use_vulkan_compute: 1
num encoder conv layers: 28
num joiner conv layers: 3
num files: 3
Processing ../test_wavs/1089-134686-0001.wav
Processing ../test_wavs/1221-135766-0001.wav
Processing ../test_wavs/1221-135766-0002.wav
Processing ../test_wavs/1089-134686-0001.wav
Processing ../test_wavs/1221-135766-0001.wav
Processing ../test_wavs/1221-135766-0002.wav
----------encoder----------
conv_15                                 : max = 15.942385        threshold = 15.930708 ␣
↪       scale = 7.972025
conv_16                                 : max = 44.978855        threshold = 17.031788 ␣
↪       scale = 7.456645
conv_17                                 : max = 17.868437        threshold = 7.830528  ␣
↪       scale = 16.218575
linear_18                               : max = 3.107259         threshold = 1.194808  ␣
↪       scale = 106.293236
linear_19                               : max = 6.193777         threshold = 4.634748  ␣
↪       scale = 27.401705
linear_20                               : max = 9.259933         threshold = 2.606617  ␣
↪       scale = 48.722160
linear_21                               : max = 5.186600         threshold = 4.790260  ␣
↪       scale = 26.512129
linear_22                               : max = 9.759041         threshold = 2.265832  ␣
↪       scale = 56.050053
linear_23                               : max = 3.931209         threshold = 3.099090  ␣
↪       scale = 40.979767
linear_24                               : max = 10.324160        threshold = 2.215561  ␣
↪       scale = 57.321835
linear_25                               : max = 3.800708         threshold = 3.599352  ␣
↪       scale = 35.284134
linear_26                               : max = 10.492444        threshold = 3.153369  ␣
↪       scale = 40.274391
linear_27                               : max = 3.660161         threshold = 2.720994  ␣
↪       scale = 46.674126
linear_28                               : max = 9.415265         threshold = 3.174434  ␣
↪       scale = 40.007133
linear_29                               : max = 4.038418         threshold = 3.118534  ␣
↪       scale = 40.724262
linear_30                               : max = 10.072084        threshold = 3.936867  ␣
↪       scale = 32.259155
linear_31                               : max = 4.342712         threshold = 3.599489  ␣
↪       scale = 35.282787
linear_32                               : max = 11.340535        threshold = 3.120308  ␣
↪       scale = 40.701103
```

(continues on next page)

(continued from previous page)

```
linear_33                                : max = 3.846987        threshold = 3.630030  ␣
↪       scale = 34.985939
linear_34                                : max = 10.686298       threshold = 2.204571  ␣
↪       scale = 57.607586
linear_35                                : max = 4.904821        threshold = 4.575518  ␣
↪       scale = 27.756420
linear_36                                : max = 11.806659       threshold = 2.585589  ␣
↪       scale = 49.118401
linear_37                                : max = 6.402340        threshold = 5.047157  ␣
↪       scale = 25.162680
linear_38                                : max = 11.174589       threshold = 1.923361  ␣
↪       scale = 66.030258
linear_39                                : max = 16.178576       threshold = 7.556058  ␣
↪       scale = 16.807705
linear_40                                : max = 12.901954       threshold = 5.301267  ␣
↪       scale = 23.956539
linear_41                                : max = 14.839805       threshold = 7.597429  ␣
↪       scale = 16.716181
linear_42                                : max = 10.178945       threshold = 2.651595  ␣
↪       scale = 47.895699
----------joiner----------
linear_2                                 : max = 24.829245       threshold = 16.627592 ␣
↪       scale = 7.637907
linear_1                                 : max = 10.746186       threshold = 5.255032  ␣
↪       scale = 24.167313
linear_3                                 : max = 1.000000        threshold = 0.999756  ␣
↪       scale = 127.031013
ncnn int8 calibration table create success, best wish for your int8 inference has a low␣
↪accuracy loss...\(^0^)/...233...
```

It generates the following two files:

```
ls -lh encoder-scale-table.txt joiner-scale-table.txt

-rw-r--r-- 1 kuangfangjun root 345K Feb 17 12:13 encoder-scale-table.txt
-rw-r--r-- 1 kuangfangjun root  17K Feb 17 12:13 joiner-scale-table.txt
```

Caution: Definitely, you need more calibration data to compute the scale table.

Finally, let us use the scale table to quantize our models into `int8`.

```
ncnn2int8

usage: ncnn2int8 [inparam] [inbin] [outparam] [outbin] [calibration table]
```

First, we quantize the encoder model:

```
cd egs/librispeech/ASR
cd icefall-asr-librispeech-lstm-transducer-stateless2-2022-09-03/exp/

ncnn2int8 \
```

(continues on next page)

(continued from previous page)

```
  ./encoder_jit_trace-pnnx.ncnn.param \
  ./encoder_jit_trace-pnnx.ncnn.bin \
  ./encoder_jit_trace-pnnx.ncnn.int8.param \
  ./encoder_jit_trace-pnnx.ncnn.int8.bin \
  ./encoder-scale-table.txt
```

Next, we quantize the joiner model:

```
ncnn2int8 \
  ./joiner_jit_trace-pnnx.ncnn.param \
  ./joiner_jit_trace-pnnx.ncnn.bin \
  ./joiner_jit_trace-pnnx.ncnn.int8.param \
  ./joiner_jit_trace-pnnx.ncnn.int8.bin \
  ./joiner-scale-table.txt
```

The above two commands generate the following 4 files:

```
-rw-r--r-- 1 kuangfangjun root 218M Feb 17 12:19 encoder_jit_trace-pnnx.ncnn.int8.bin
-rw-r--r-- 1 kuangfangjun root  21K Feb 17 12:19 encoder_jit_trace-pnnx.ncnn.int8.param
-rw-r--r-- 1 kuangfangjun root 774K Feb 17 12:19 joiner_jit_trace-pnnx.ncnn.int8.bin
-rw-r--r-- 1 kuangfangjun root  496 Feb 17 12:19 joiner_jit_trace-pnnx.ncnn.int8.param
```

Congratulations! You have successfully quantized your model from `float32` to `int8`.

Caution: `ncnn.int8.param` and `ncnn.int8.bin` must be used in pairs.

You can replace `ncnn.param` and `ncnn.bin` with `ncnn.int8.param` and `ncnn.int8.bin` in sherpa-ncnn if you like.

For instance, to use only the `int8` encoder in `sherpa-ncnn`, you can replace the following invocation:

```
cd egs/librispeech/ASR
cd icefall-asr-librispeech-lstm-transducer-stateless2-2022-09-03/exp/

sherpa-ncnn \
  ../data/lang_bpe_500/tokens.txt \
  ./encoder_jit_trace-pnnx.ncnn.param \
  ./encoder_jit_trace-pnnx.ncnn.bin \
  ./decoder_jit_trace-pnnx.ncnn.param \
  ./decoder_jit_trace-pnnx.ncnn.bin \
  ./joiner_jit_trace-pnnx.ncnn.param \
  ./joiner_jit_trace-pnnx.ncnn.bin \
  ../test_wavs/1089-134686-0001.wav
```

with

```
cd egs/librispeech/ASR
cd icefall-asr-librispeech-conv-emformer-transducer-stateless2-2022-07-05/
↪exp/

sherpa-ncnn \
  ../data/lang_bpe_500/tokens.txt \
  ./encoder_jit_trace-pnnx.ncnn.int8.param \
  ./encoder_jit_trace-pnnx.ncnn.int8.bin \
  ./decoder_jit_trace-pnnx.ncnn.param \
  ./decoder_jit_trace-pnnx.ncnn.bin \
  ./joiner_jit_trace-pnnx.ncnn.param \
  ./joiner_jit_trace-pnnx.ncnn.bin \
  ../test_wavs/1089-134686-0001.wav
```

The following table compares again the file sizes:

File name	File size
encoder_jit_trace-pnnx.pt	318 MB
decoder_jit_trace-pnnx.pt	1010 KB
joiner_jit_trace-pnnx.pt	3.0 MB
encoder_jit_trace-pnnx.ncnn.bin (fp16)	159 MB
decoder_jit_trace-pnnx.ncnn.bin (fp16)	503 KB
joiner_jit_trace-pnnx.ncnn.bin (fp16)	1.5 MB
encoder_jit_trace-pnnx.ncnn.bin (fp32)	317 MB
joiner_jit_trace-pnnx.ncnn.bin (fp32)	3.0 MB
encoder_jit_trace-pnnx.ncnn.int8.bin	218 MB
joiner_jit_trace-pnnx.ncnn.int8.bin	774 KB

You can see that the file size of the joiner model after `int8` quantization is much smaller. However, the size of the encoder model is even larger than the `fp16` counterpart. The reason is that ncnn currently does not support quantizing `LSTM` layers into `8-bit`. Please see https://github.com/Tencent/ncnn/issues/4532

Hint: Currently, only linear layers and convolutional layers are quantized with `int8`, so you don't see an exact `4x` reduction in file sizes.

Note: You need to test the recognition accuracy after `int8` quantization.

That's it! Have fun with sherpa-ncnn!

CHAPTER

SIX

RECIPES

This page contains various recipes in `icefall`. Currently, only speech recognition recipes are provided.

We may add recipes for other tasks as well in the future.

6.1 Non Streaming ASR

6.1.1 aishell

Aishell is an open-source Chinese Mandarin speech corpus published by Beijing Shell Shell Technology Co.,Ltd.

400 people from different accent areas in China are invited to participate in the recording, which is conducted in a quiet indoor environment using high fidelity microphone and downsampled to 16kHz. The manual transcription accuracy is above 95%, through professional speech annotation and strict quality inspection. The data is free for academic use. We hope to provide moderate amount of data for new researchers in the field of speech recognition.

It can be downloaded from https://www.openslr.org/33/

TDNN-LSTM CTC

This tutorial shows you how to run a tdnn-lstm ctc model with the Aishell dataset.

Hint: We assume you have read the page *Installation* and have setup the environment for `icefall`.

Hint: We recommend you to use a GPU or several GPUs to run this recipe.

In this tutorial, you will learn:

- (1) How to prepare data for training and decoding
- (2) How to start the training, either with a single GPU or multiple GPUs
- (3) How to do decoding after training.
- (4) How to use a pre-trained model, provided by us

Data preparation

```
$ cd egs/aishell/ASR
$ ./prepare.sh
```

The script `./prepare.sh` handles the data preparation for you, **automagically**. All you need to do is to run it.

The data preparation contains several stages, you can use the following two options:

- `--stage`
- `--stop-stage`

to control which stage(s) should be run. By default, all stages are executed.

For example,

```
$ cd egs/aishell/ASR
$ ./prepare.sh --stage 0 --stop-stage 0
```

means to run only stage 0.

To run stage 2 to stage 5, use:

```
$ ./prepare.sh --stage 2 --stop-stage 5
```

Hint: If you have pre-downloaded the Aishell dataset and the musan dataset, say, they are saved in `/tmp/aishell` and `/tmp/musan`, you can modify the `dl_dir` variable in `./prepare.sh` to point to `/tmp` so that `./prepare.sh` won't re-download them.

Hint: A 3-gram language model will be downloaded from huggingface, we assume you have intalled and initialized `git-lfs`. If not, you could install `git-lfs` by

```
$ sudo apt-get install git-lfs
$ git-lfs install
```

If you don't have the `sudo` permission, you could download the git-lfs binary here, then add it to you `PATH`.

Note: All generated files by `./prepare.sh`, e.g., features, lexicon, etc, are saved in `./data` directory.

Training

Configurable options

```
$ cd egs/aishell/ASR
$ ./tdnn_lstm_ctc/train.py --help
```

shows you the training options that can be passed from the commandline. The following options are used quite often:

- `--num-epochs`

  It is the number of epochs to train. For instance, `./tdnn_lstm_ctc/train.py --num-epochs 30` trains for 30 epochs and generates `epoch-0.pt`, `epoch-1.pt`, . . . , `epoch-29.pt` in the folder `./tdnn_lstm_ctc/exp`.

- `--start-epoch`

  It's used to resume training. `./tdnn_lstm_ctc/train.py --start-epoch 10` loads the checkpoint `./tdnn_lstm_ctc/exp/epoch-9.pt` and starts training from epoch 10, based on the state from epoch 9.

- `--world-size`

  It is used for multi-GPU single-machine DDP training.

  - (a) If it is 1, then no DDP training is used.
  - (b) If it is 2, then GPU 0 and GPU 1 are used for DDP training.

  The following shows some use cases with it.

  Use case 1: You have 4 GPUs, but you only want to use GPU 0 and GPU 2 for training. You can do the following:

  ```
  $ cd egs/aishell/ASR
  $ export CUDA_VISIBLE_DEVICES="0,2"
  $ ./tdnn_lstm_ctc/train.py --world-size 2
  ```

  Use case 2: You have 4 GPUs and you want to use all of them for training. You can do the following:

  ```
  $ cd egs/aishell/ASR
  $ ./tdnn_lstm_ctc/train.py --world-size 4
  ```

  Use case 3: You have 4 GPUs but you only want to use GPU 3 for training. You can do the following:

  ```
  $ cd egs/aishell/ASR
  $ export CUDA_VISIBLE_DEVICES="3"
  $ ./tdnn_lstm_ctc/train.py --world-size 1
  ```

  Caution: Only multi-GPU single-machine DDP training is implemented at present. Multi-GPU multi-machine DDP training will be added later.

- `--max-duration`

  It specifies the number of seconds over all utterances in a batch, before **padding**. If you encounter CUDA OOM, please reduce it. For instance, if your are using V100 NVIDIA GPU, we recommend you to set it to `2000`.

  Hint: Due to padding, the number of seconds of all utterances in a batch will usually be larger than `--max-duration`.

  A larger value for `--max-duration` may cause OOM during training, while a smaller value may increase the training time. You have to tune it.

Pre-configured options

There are some training options, e.g., weight decay, number of warmup steps, results dir, etc, that are not passed from the commandline. They are pre-configured by the function `get_params()` in tdnn_lstm_ctc/train.py

You don't need to change these pre-configured parameters. If you really need to change them, please modify `./tdnn_lstm_ctc/train.py` directly.

> **Caution:** The training set is perturbed by speed with two factors: 0.9 and 1.1. Each epoch actually processes `3x150 == 450` hours of data.

Training logs

Training logs and checkpoints are saved in `tdnn_lstm_ctc/exp`. You will find the following files in that directory:

- `epoch-0.pt`, `epoch-1.pt`, ...

  These are checkpoint files, containing model `state_dict` and optimizer `state_dict`. To resume training from some checkpoint, say `epoch-10.pt`, you can use:

  ```
  $ ./tdnn_lstm_ctc/train.py --start-epoch 11
  ```

- `tensorboard/`

  This folder contains TensorBoard logs. Training loss, validation loss, learning rate, etc, are recorded in these logs. You can visualize them by:

  ```
  $ cd tdnn_lstm_ctc/exp/tensorboard
  $ tensorboard dev upload --logdir . --description "TDNN-LSTM CTC training␣
  ↪for Aishell with icefall"
  ```

  It will print something like below:

  ```
  TensorFlow installation not found - running with reduced feature set.
  Upload started and will continue reading any new data as it's added to the␣
  ↪logdir.

  To stop uploading, press Ctrl-C.

  New experiment created. View your TensorBoard at: https://tensorboard.dev/
  ↪experiment/LJI9MWUORLOw3jkdhxwk8A/

  [2021-09-13T11:59:23] Started scanning logdir.
  [2021-09-13T11:59:24] Total uploaded: 4454 scalars, 0 tensors, 0 binary␣
  ↪objects
  Listening for new data in logdir...
  ```

  Note there is a URL in the above output, click it and you will see the following screenshot:

- `log/log-train-xxxx`

  It is the detailed training log in text format, same as the one you saw printed to the console during training.

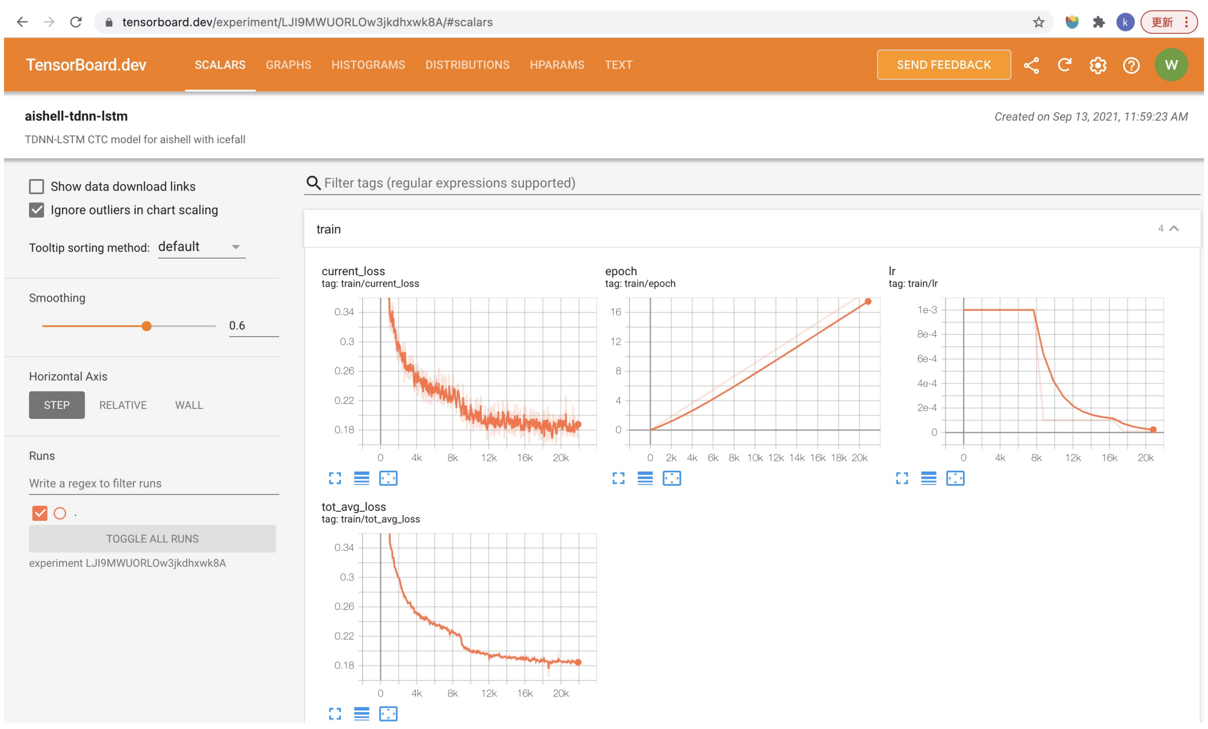

tensorboard.dev/experiment/LJI9MWUORLOw3jkdhxwk8A/#scalars
TensorBoard.dev
SCALARS
GRAPHS
HISTOGRAMS
DISTRIBUTIONS
HPARAMS
TEXT
SEND FEEDBACK
aishell-tdnn-lstm
TDNN-LSTM CTC model for aishell with icefall
Created on Sep 13, 2021, 11:59:23 AM
Show data download links
Ignore outliers in chart scaling
Tooltip sorting method: default
Smoothing 0.6
Horizontal Axis
STEP
RELATIVE
WALL
Runs
Write a regex to filter runs
TOGGLE ALL RUNS
experiment LJI9MWUORLOw3jkdhxwk8A
Filter tags (regular expressions supported)
train
current_loss
tag: train/current_loss
epoch
tag: train/epoch
lr
tag: train/lr
tot_avg_loss
tag: train/tot_avg_loss

Fig. 6.1: TensorBoard screenshot.

Usage examples

The following shows typical use cases:

Case 1

```
$ cd egs/aishell/ASR
$ export CUDA_VISIBLE_DEVICES="0,3"
$ ./tdnn_lstm_ctc/train.py --world-size 2
```

It uses GPU 0 and GPU 3 for DDP training.

Case 2

```
$ cd egs/aishell/ASR
$ ./tdnn_lstm_ctc/train.py --num-epochs 10 --start-epoch 3
```

It loads checkpoint `./tdnn_lstm_ctc/exp/epoch-2.pt` and starts training from epoch 3. Also, it trains for 10 epochs.

Decoding

The decoding part uses checkpoints saved by the training part, so you have to run the training part first.

```
$ cd egs/aishell/ASR
$ ./tdnn_lstm_ctc/decode.py --help
```

shows the options for decoding.

The commonly used options are:

- `--method`

  This specifies the decoding method.

  The following command uses attention decoder for rescoring:

  ```
  $ cd egs/aishell/ASR
  $ ./tdnn_lstm_ctc/decode.py --method 1best --max-duration 100
  ```

- `--max-duration`

  It has the same meaning as the one during training. A larger value may cause OOM.

Pre-trained Model

We have uploaded a pre-trained model to https://huggingface.co/pkufool/icefall_asr_aishell_tdnn_lstm_ctc.

We describe how to use the pre-trained model to transcribe a sound file or multiple sound files in the following.

Install kaldifeat

kaldifeat is used to extract features for a single sound file or multiple sound files at the same time.

Please refer to https://github.com/csukuangfj/kaldifeat for installation.

Download the pre-trained model

The following commands describe how to download the pre-trained model:

```
$ cd egs/aishell/ASR
$ mkdir tmp
$ cd tmp
$ git lfs install
$ git clone https://huggingface.co/pkufool/icefall_asr_aishell_tdnn_lstm_ctc
```

Caution: You have to use `git lfs` to download the pre-trained model.

Caution: In order to use this pre-trained model, your k2 version has to be v1.7 or later.

After downloading, you will have the following files:

```
$ cd egs/aishell/ASR
$ tree tmp
```

```
tmp/
`-- icefall_asr_aishell_tdnn_lstm_ctc
    |-- README.md
    |-- data
    |   `-- lang_phone
    |       |-- HLG.pt
    |       |-- tokens.txt
    |       `-- words.txt
    |-- exp
    |   `-- pretrained.pt
    `-- test_waves
        |-- BAC009S0764W0121.wav
        |-- BAC009S0764W0122.wav
        |-- BAC009S0764W0123.wav
        `-- trans.txt

5 directories, 9 files
```

File descriptions:

- `data/lang_phone/HLG.pt`

  It is the decoding graph.

- `data/lang_phone/tokens.txt`

  It contains tokens and their IDs. Provided only for convenience so that you can look up the SOS/EOS ID easily.

- `data/lang_phone/words.txt`

  It contains words and their IDs.

- `exp/pretrained.pt`

  It contains pre-trained model parameters, obtained by averaging checkpoints from `epoch-18.pt` to `epoch-40.pt`. Note: We have removed optimizer `state_dict` to reduce file size.

- `test_waves/*.wav`

  It contains some test sound files from Aishell `test` dataset.

- `test_waves/trans.txt`

  It contains the reference transcripts for the sound files in *test_waves/*.

The information of the test sound files is listed below:

```
$ soxi tmp/icefall_asr_aishell_tdnn_lstm_ctc/test_waves/*.wav

Input File     : 'tmp/icefall_asr_aishell_tdnn_lstm_ctc/test_waves/BAC009S0764W0121.wav'
Channels       : 1
Sample Rate    : 16000
Precision      : 16-bit
Duration       : 00:00:04.20 = 67263 samples ~ 315.295 CDDA sectors
File Size      : 135k
```

(continues on next page)

(continued from previous page)

```
Bit Rate       : 256k
Sample Encoding: 16-bit Signed Integer PCM


Input File     : 'tmp/icefall_asr_aishell_tdnn_lstm_ctc/test_waves/BAC009S0764W0122.wav'
Channels       : 1
Sample Rate    : 16000
Precision      : 16-bit
Duration       : 00:00:04.12 = 65840 samples ~ 308.625 CDDA sectors
File Size      : 132k
Bit Rate       : 256k
Sample Encoding: 16-bit Signed Integer PCM


Input File     : 'tmp/icefall_asr_aishell_tdnn_lstm_ctc/test_waves/BAC009S0764W0123.wav'
Channels       : 1
Sample Rate    : 16000
Precision      : 16-bit
Duration       : 00:00:04.00 = 64000 samples ~ 300 CDDA sectors
File Size      : 128k
Bit Rate       : 256k
Sample Encoding: 16-bit Signed Integer PCM

Total Duration of 3 files: 00:00:12.32
```

Usage

```
$ cd egs/aishell/ASR
$ ./tdnn_lstm_ctc/pretrained.py --help
```

displays the help information.

HLG decoding

HLG decoding uses the best path of the decoding lattice as the decoding result.

The command to run HLG decoding is:

```
$ cd egs/aishell/ASR
$ ./tdnn_lstm_ctc/pretrained.py \
  --checkpoint ./tmp/icefall_asr_aishell_tdnn_lstm_ctc/exp/pretrained.pt \
  --words-file ./tmp/icefall_asr_aishell_tdnn_lstm_ctc/data/lang_phone/words.txt \
  --HLG ./tmp/icefall_asr_aishell_tdnn_lstm_ctc/data/lang_phone/HLG.pt \
  --method 1best \
  ./tmp/icefall_asr_aishell_tdnn_lstm_ctc/test_waves/BAC009S0764W0121.wav \
  ./tmp/icefall_asr_aishell_tdnn_lstm_ctc/test_waves/BAC009S0764W0122.wav \
  ./tmp/icefall_asr_aishell_tdnn_lstm_ctc/test_waves/BAC009S0764W0123.wav
```

The output is given below:

```
2021-09-13 15:00:55,858 INFO [pretrained.py:140] device: cuda:0
2021-09-13 15:00:55,858 INFO [pretrained.py:142] Creating model
2021-09-13 15:01:05,389 INFO [pretrained.py:154] Loading HLG from ./tmp/icefall_asr_
↪aishell_tdnn_lstm_ctc/data/lang_phone/HLG.pt
2021-09-13 15:01:06,531 INFO [pretrained.py:161] Constructing Fbank computer
2021-09-13 15:01:06,536 INFO [pretrained.py:171] Reading sound files: ['./tmp/icefall_
↪asr_aishell_tdnn_lstm_ctc/test_waves/BAC009S0764W0121.wav', './tmp/icefall_asr_aishell_
↪tdnn_lstm_ctc/test_waves/BAC009S0764W0122.wav', './tmp/icefall_asr_aishell_tdnn_lstm_
↪ctc/test_waves/BAC009S0764W0123.wav']
2021-09-13 15:01:06,539 INFO [pretrained.py:177] Decoding started
2021-09-13 15:01:06,917 INFO [pretrained.py:207] Use HLG decoding
2021-09-13 15:01:07,129 INFO [pretrained.py:220]
./tmp/icefall_asr_aishell_tdnn_lstm_ctc/test_waves/BAC009S0764W0121.wav:


./tmp/icefall_asr_aishell_tdnn_lstm_ctc/test_waves/BAC009S0764W0122.wav:


./tmp/icefall_asr_aishell_tdnn_lstm_ctc/test_waves/BAC009S0764W0123.wav:



2021-09-13 15:01:07,129 INFO [pretrained.py:222] Decoding Done
```

Colab notebook

We do provide a colab notebook for this recipe showing how to use a pre-trained model.

Congratulations! You have finished the aishell ASR recipe with TDNN-LSTM CTC models in `icefall`.

Conformer CTC

This tutorial shows you how to run a conformer ctc model with the Aishell dataset.

Hint: We assume you have read the page *Installation* and have setup the environment for `icefall`.

Hint: We recommend you to use a GPU or several GPUs to run this recipe.

In this tutorial, you will learn:

- (1) How to prepare data for training and decoding
- (2) How to start the training, either with a single GPU or multiple GPUs
- (3) How to do decoding after training, with ctc-decoding, 1best and attention decoder rescoring
- (4) How to use a pre-trained model, provided by us

Data preparation

```
$ cd egs/aishell/ASR
$ ./prepare.sh
```

The script `./prepare.sh` handles the data preparation for you, **automagically**. All you need to do is to run it.

The data preparation contains several stages, you can use the following two options:

- `--stage`
- `--stop-stage`

to control which stage(s) should be run. By default, all stages are executed.

For example,

```
$ cd egs/aishell/ASR
$ ./prepare.sh --stage 0 --stop-stage 0
```

means to run only stage 0.

To run stage 2 to stage 5, use:

```
$ ./prepare.sh --stage 2 --stop-stage 5
```

Hint: If you have pre-downloaded the Aishell dataset and the musan dataset, say, they are saved in `/tmp/aishell` and `/tmp/musan`, you can modify the `dl_dir` variable in `./prepare.sh` to point to `/tmp` so that `./prepare.sh` won't re-download them.

Hint: A 3-gram language model will be downloaded from huggingface, we assume you have intalled and initialized `git-lfs`. If not, you could install `git-lfs` by

```
$ sudo apt-get install git-lfs
$ git-lfs install
```

If you don't have the `sudo` permission, you could download the git-lfs binary here, then add it to you `PATH`.

Note: All generated files by `./prepare.sh`, e.g., features, lexicon, etc, are saved in `./data` directory.

Training

Configurable options

```
$ cd egs/aishell/ASR
$ ./conformer_ctc/train.py --help
```

shows you the training options that can be passed from the commandline. The following options are used quite often:

- `--exp-dir`

  The experiment folder to save logs and model checkpoints, default `./conformer_ctc/exp`.

- `--num-epochs`

  It is the number of epochs to train. For instance, `./conformer_ctc/train.py --num-epochs 30` trains for 30 epochs and generates `epoch-0.pt`, `epoch-1.pt`, . . . , `epoch-29.pt` in the folder set by `--exp-dir`.

- `--start-epoch`

  It's used to resume training. `./conformer_ctc/train.py --start-epoch 10` loads the checkpoint `./conformer_ctc/exp/epoch-9.pt` and starts training from epoch 10, based on the state from epoch 9.

- `--world-size`

  It is used for multi-GPU single-machine DDP training.

  - (a) If it is 1, then no DDP training is used.
  - (b) If it is 2, then GPU 0 and GPU 1 are used for DDP training.

  The following shows some use cases with it.

  Use case 1: You have 4 GPUs, but you only want to use GPU 0 and GPU 2 for training. You can do the following:

  ```
  $ cd egs/aishell/ASR
  $ export CUDA_VISIBLE_DEVICES="0,2"
  $ ./conformer_ctc/train.py --world-size 2
  ```

  Use case 2: You have 4 GPUs and you want to use all of them for training. You can do the following:

  ```
  $ cd egs/aishell/ASR
  $ ./conformer_ctc/train.py --world-size 4
  ```

  Use case 3: You have 4 GPUs but you only want to use GPU 3 for training. You can do the following:

  ```
  $ cd egs/aishell/ASR
  $ export CUDA_VISIBLE_DEVICES="3"
  $ ./conformer_ctc/train.py --world-size 1
  ```

  Caution: Only multi-GPU single-machine DDP training is implemented at present. Multi-GPU multi-machine DDP training will be added later.

- `--max-duration`

  It specifies the number of seconds over all utterances in a batch, before **padding**. If you encounter CUDA OOM, please reduce it. For instance, if your are using V100 NVIDIA GPU, we recommend you to set it to `200`.

  Hint: Due to padding, the number of seconds of all utterances in a batch will usually be larger than `--max-duration`.

  A larger value for `--max-duration` may cause OOM during training, while a smaller value may increase the training time. You have to tune it.

Pre-configured options

There are some training options, e.g., weight decay, number of warmup steps, etc, that are not passed from the commandline. They are pre-configured by the function `get_params()` in conformer_ctc/train.py

You don't need to change these pre-configured parameters. If you really need to change them, please modify `./conformer_ctc/train.py` directly.

Caution: The training set is perturbed by speed with two factors: 0.9 and 1.1. Each epoch actually processes `3x150 == 450` hours of data.

Training logs

Training logs and checkpoints are saved in the folder set by `--exp-dir` (default `conformer_ctc/exp`). You will find the following files in that directory:

- `epoch-0.pt`, `epoch-1.pt`, . . .

  These are checkpoint files, containing model `state_dict` and optimizer `state_dict`. To resume training from some checkpoint, say `epoch-10.pt`, you can use:

  ```
  $ ./conformer_ctc/train.py --start-epoch 11
  ```

- `tensorboard/`

  This folder contains TensorBoard logs. Training loss, validation loss, learning rate, etc, are recorded in these logs. You can visualize them by:

  ```
  $ cd conformer_ctc/exp/tensorboard
  $ tensorboard dev upload --logdir . --name "Aishell conformer ctc training␣
  ↪with icefall" --description "Training with new LabelSmoothing loss, see␣
  ↪https://github.com/k2-fsa/icefall/pull/109"
  ```

  It will print something like below:

  ```
  TensorFlow installation not found - running with reduced feature set.
  Upload started and will continue reading any new data as it's added to the␣
  ↪logdir.

  To stop uploading, press Ctrl-C.

  New experiment created. View your TensorBoard at: https://tensorboard.dev/
  ↪experiment/engw8KSkTZqS24zBV5dgCg/

  [2021-11-22T11:09:27] Started scanning logdir.
  [2021-11-22T11:10:14] Total uploaded: 116068 scalars, 0 tensors, 0 binary␣
  ↪objects
  Listening for new data in logdir...
  ```

  Note there is a URL in the above output, click it and you will see the following screenshot:

- `log/log-train-xxxx`

  It is the detailed training log in text format, same as the one you saw printed to the console during training.

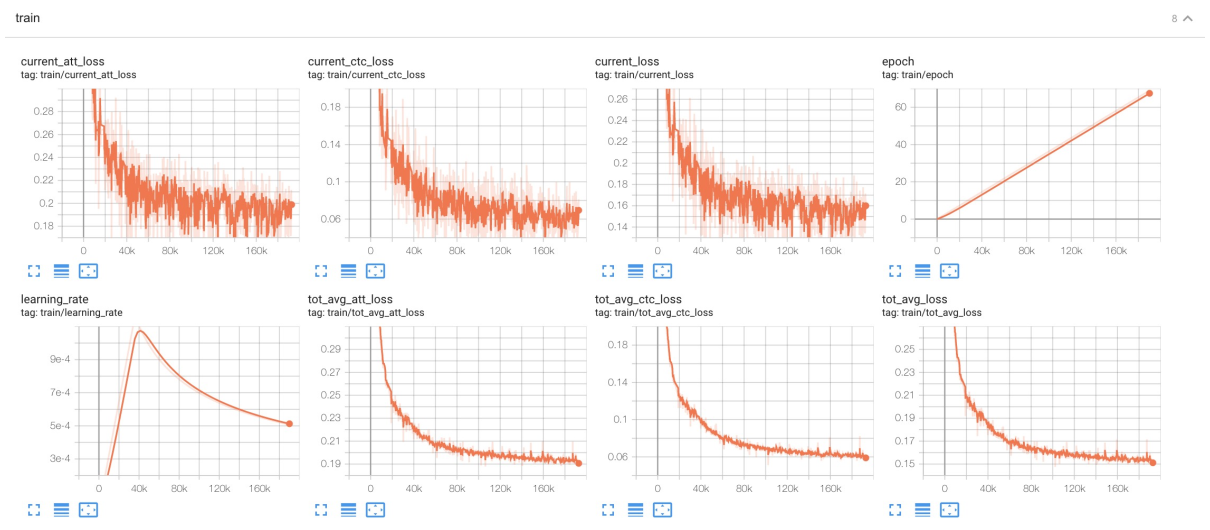


Fig. 6.2: TensorBoard screenshot.

Usage examples

The following shows typical use cases:

Case 1

```
$ cd egs/aishell/ASR
$ ./conformer_ctc/train.py --max-duration 200
```

It uses `--max-duration` of 200 to avoid OOM.

Case 2

```
$ cd egs/aishell/ASR
$ export CUDA_VISIBLE_DEVICES="0,3"
$ ./conformer_ctc/train.py --world-size 2
```

It uses GPU 0 and GPU 3 for DDP training.

Case 3

```
$ cd egs/aishell/ASR
$ ./conformer_ctc/train.py --num-epochs 10 --start-epoch 3
```

It loads checkpoint `./conformer_ctc/exp/epoch-2.pt` and starts training from epoch 3. Also, it trains for 10 epochs.

Decoding

The decoding part uses checkpoints saved by the training part, so you have to run the training part first.

```
$ cd egs/aishell/ASR
$ ./conformer_ctc/decode.py --help
```

shows the options for decoding.

The commonly used options are:

- `--method`

  This specifies the decoding method.

  The following command uses attention decoder for rescoring:

  ```
  $ cd egs/aishell/ASR
  $ ./conformer_ctc/decode.py --method attention-decoder --max-duration 30 --nbest-
  ↪scale 0.5
  ```

- `--nbest-scale`

  It is used to scale down lattice scores so that there are more unique paths for rescoring.

- `--max-duration`

  It has the same meaning as the one during training. A larger value may cause OOM.

Pre-trained Model

We have uploaded a pre-trained model to https://huggingface.co/pkufool/icefall_asr_aishell_conformer_ctc.

We describe how to use the pre-trained model to transcribe a sound file or multiple sound files in the following.

Install kaldifeat

kaldifeat is used to extract features for a single sound file or multiple sound files at the same time.

Please refer to https://github.com/csukuangfj/kaldifeat for installation.

Download the pre-trained model

The following commands describe how to download the pre-trained model:

```
$ cd egs/aishell/ASR
$ mkdir tmp
$ cd tmp
$ git lfs install
$ git clone https://huggingface.co/pkufool/icefall_asr_aishell_conformer_ctc
```

Caution: You have to use `git lfs` to download the pre-trained model.

Caution: In order to use this pre-trained model, your k2 version has to be v1.7 or later.

After downloading, you will have the following files:

```
$ cd egs/aishell/ASR
$ tree tmp
```

```
tmp/
`-- icefall_asr_aishell_conformer_ctc
    |-- README.md
    |-- data
    |   `-- lang_char
    |       |-- HLG.pt
    |       |-- tokens.txt
    |       `-- words.txt
    |-- exp
    |   `-- pretrained.pt
    `-- test_waves
        |-- BAC009S0764W0121.wav
        |-- BAC009S0764W0122.wav
        |-- BAC009S0764W0123.wav
        `-- trans.txt

5 directories, 9 files
```

File descriptions:

- `data/lang_char/HLG.pt`

  It is the decoding graph.

- `data/lang_char/tokens.txt`

  It contains tokens and their IDs. Provided only for convenience so that you can look up the SOS/EOS ID easily.

- `data/lang_char/words.txt`

  It contains words and their IDs.

- `exp/pretrained.pt`

  It contains pre-trained model parameters, obtained by averaging checkpoints from `epoch-25.pt` to `epoch-84.pt`. Note: We have removed optimizer `state_dict` to reduce file size.

- `test_waves/*.wav`

  It contains some test sound files from Aishell `test` dataset.

- `test_waves/trans.txt`

  It contains the reference transcripts for the sound files in *test_waves/*.

The information of the test sound files is listed below:

```
$ soxi tmp/icefall_asr_aishell_conformer_ctc/test_waves/*.wav

Input File     : 'tmp/icefall_asr_aishell_conformer_ctc/test_waves/BAC009S0764W0121.wav'
```

(continues on next page)

(continued from previous page)

```
Channels       : 1
Sample Rate    : 16000
Precision      : 16-bit
Duration       : 00:00:04.20 = 67263 samples ~ 315.295 CDDA sectors
File Size      : 135k
Bit Rate       : 256k
Sample Encoding: 16-bit Signed Integer PCM


Input File     : 'tmp/icefall_asr_aishell_conformer_ctc/test_waves/BAC009S0764W0122.wav'
Channels       : 1
Sample Rate    : 16000
Precision      : 16-bit
Duration       : 00:00:04.12 = 65840 samples ~ 308.625 CDDA sectors
File Size      : 132k
Bit Rate       : 256k
Sample Encoding: 16-bit Signed Integer PCM


Input File     : 'tmp/icefall_asr_aishell_conformer_ctc/test_waves/BAC009S0764W0123.wav'
Channels       : 1
Sample Rate    : 16000
Precision      : 16-bit
Duration       : 00:00:04.00 = 64000 samples ~ 300 CDDA sectors
File Size      : 128k
Bit Rate       : 256k
Sample Encoding: 16-bit Signed Integer PCM

Total Duration of 3 files: 00:00:12.32
```

Usage

```
$ cd egs/aishell/ASR
$ ./conformer_ctc/pretrained.py --help
```

displays the help information.

It supports three decoding methods:

- CTC decoding
- HLG decoding
- HLG + attention decoder rescoring

CTC decoding

CTC decoding only uses the ctc topology for decoding without a lexicon and language model

The command to run CTC decoding is:

```
$ cd egs/aishell/ASR
$ ./conformer_ctc/pretrained.py \
  --checkpoint ./tmp/icefall_asr_aishell_conformer_ctc/exp/pretrained.pt \
  --tokens-file ./tmp/icefall_asr_aishell_conformer_ctc/data/lang_char/tokens.txt \
  --method ctc-decoding \
  ./tmp/icefall_asr_aishell_conformer_ctc/test_waves/BAC009S0764W0121.wav \
  ./tmp/icefall_asr_aishell_conformer_ctc/test_waves/BAC009S0764W0122.wav \
  ./tmp/icefall_asr_aishell_conformer_ctc/test_waves/BAC009S0764W0123.wav
```

The output is given below:

```
2021-11-18 07:53:41,707 INFO [pretrained.py:229] {'sample_rate': 16000, 'subsampling_
↪factor': 4, 'feature_dim': 80, 'nhead': 4, 'attention_dim': 512, 'num_decoder_layers':␣
↪6, 'vgg_frontend': False, 'use_feat_batchnorm': True, 'search_beam': 20, 'output_beam
↪': 8, 'min_active_states': 30, 'max_active_states': 10000, 'use_double_scores': True,
↪'env_info': {'k2-version': '1.9', 'k2-build-type': 'Release', 'k2-with-cuda': True,
↪'k2-git-sha1': 'f2fd997f752ed11bbef4c306652c433e83f9cf12', 'k2-git-date': 'Sun Sep 19␣
↪09:41:46 2021', 'lhotse-version': '0.11.0.dev+git.33cfe45.clean', 'torch-cuda-available
↪': True, 'torch-cuda-version': '10.1', 'python-version': '3.8', 'icefall-git-branch':
↪'aishell', 'icefall-git-sha1': 'd57a873-dirty', 'icefall-git-date': 'Wed Nov 17␣
↪19:53:25 2021', 'icefall-path': '/ceph-hw/kangwei/code/icefall_aishell3', 'k2-path': '/
↪ceph-hw/kangwei/code/k2_release/k2/k2/python/k2/__init__.py', 'lhotse-path': '/ceph-hw/
↪kangwei/code/lhotse/lhotse/__init__.py'}, 'checkpoint': './tmp/icefall_asr_aishell_
↪conformer_ctc/exp/pretrained.pt', 'tokens_file': './tmp/icefall_asr_aishell_conformer_
↪ctc/data/lang_char/tokens.txt', 'words_file': None, 'HLG': None, 'method': 'ctc-
↪decoding', 'num_paths': 100, 'ngram_lm_scale': 0.3, 'attention_decoder_scale': 0.9,
↪'nbest_scale': 0.5, 'sos_id': 1, 'eos_id': 1, 'num_classes': 4336, 'sound_files': ['./
↪tmp/icefall_asr_aishell_conformer_ctc/test_waves/BAC009S0764W0121.wav', './tmp/icefall_
↪asr_aishell_conformer_ctc/test_waves/BAC009S0764W0122.wav', './tmp/icefall_asr_aishell_
↪conformer_ctc/test_waves/BAC009S0764W0123.wav']}
2021-11-18 07:53:41,708 INFO [pretrained.py:240] device: cuda:0
2021-11-18 07:53:41,708 INFO [pretrained.py:242] Creating model
2021-11-18 07:53:51,131 INFO [pretrained.py:259] Constructing Fbank computer
2021-11-18 07:53:51,134 INFO [pretrained.py:269] Reading sound files: ['./tmp/icefall_
↪asr_aishell_conformer_ctc/test_waves/BAC009S0764W0121.wav', './tmp/icefall_asr_aishell_
↪conformer_ctc/test_waves/BAC009S0764W0122.wav', './tmp/icefall_asr_aishell_conformer_
↪ctc/test_waves/BAC009S0764W0123.wav']
2021-11-18 07:53:51,138 INFO [pretrained.py:275] Decoding started
2021-11-18 07:53:51,241 INFO [pretrained.py:293] Use CTC decoding
2021-11-18 07:53:51,704 INFO [pretrained.py:369]
./tmp/icefall_asr_aishell_conformer_ctc/test_waves/BAC009S0764W0121.wav:


./tmp/icefall_asr_aishell_conformer_ctc/test_waves/BAC009S0764W0122.wav:


./tmp/icefall_asr_aishell_conformer_ctc/test_waves/BAC009S0764W0123.wav:

```

(continues on next page)

(continued from previous page)

```


2021-11-18 07:53:51,704 INFO [pretrained.py:371] Decoding Done
```

HLG decoding

HLG decoding uses the best path of the decoding lattice as the decoding result.

The command to run HLG decoding is:

```
$ cd egs/aishell/ASR
$ ./conformer_ctc/pretrained.py \
  --checkpoint ./tmp/icefall_asr_aishell_conformer_ctc/exp/pretrained.pt \
  --words-file ./tmp/icefall_asr_aishell_conformer_ctc/data/lang_char/words.txt \
  --HLG ./tmp/icefall_asr_aishell_conformer_ctc/data/lang_char/HLG.pt \
  --method 1best \
  ./tmp/icefall_asr_aishell_conformer_ctc/test_waves/BAC009S0764W0121.wav \
  ./tmp/icefall_asr_aishell_conformer_ctc/test_waves/BAC009S0764W0122.wav \
  ./tmp/icefall_asr_aishell_conformer_ctc/test_waves/BAC009S0764W0123.wav
```

The output is given below:

```
2021-11-18 07:37:38,683 INFO [pretrained.py:229] {'sample_rate': 16000, 'subsampling_
↪factor': 4, 'feature_dim': 80, 'nhead': 4, 'attention_dim': 512, 'num_decoder_layers':␣
↪6, 'vgg_frontend': False, 'use_feat_batchnorm': True, 'search_beam': 20, 'output_beam
↪': 8, 'min_active_states': 30, 'max_active_states': 10000, 'use_double_scores': True,
↪'env_info': {'k2-version': '1.9', 'k2-build-type': 'Release', 'k2-with-cuda': True,
↪'k2-git-sha1': 'f2fd997f752ed11bbef4c306652c433e83f9cf12', 'k2-git-date': 'Sun Sep 19␣
↪09:41:46 2021', 'lhotse-version': '0.11.0.dev+git.33cfe45.clean', 'torch-cuda-available
↪': True, 'torch-cuda-version': '10.1', 'python-version': '3.8', 'icefall-git-branch':
↪'aishell', 'icefall-git-sha1': 'd57a873-clean', 'icefall-git-date': 'Wed Nov 17␣
↪19:53:25 2021', 'icefall-path': '/ceph-hw/kangwei/code/icefall_aishell3', 'k2-path': '/
↪ceph-hw/kangwei/code/k2_release/k2/k2/python/k2/__init__.py', 'lhotse-path': '/ceph-hw/
↪kangwei/code/lhotse/lhotse/__init__.py'}, 'checkpoint': './tmp/icefall_asr_aishell_
↪conformer_ctc/exp/pretrained.pt', 'tokens_file': None, 'words_file': './tmp/icefall_
↪asr_aishell_conformer_ctc/data/lang_char/words.txt', 'HLG': './tmp/icefall_asr_aishell_
↪conformer_ctc/data/lang_char/HLG.pt', 'method': '1best', 'num_paths': 100, 'ngram_lm_
↪scale': 0.3, 'attention_decoder_scale': 0.9, 'nbest_scale': 0.5, 'sos_id': 1, 'eos_id
↪': 1, 'num_classes': 4336, 'sound_files': ['./tmp/icefall_asr_aishell_conformer_ctc/
↪test_waves/BAC009S0764W0121.wav', './tmp/icefall_asr_aishell_conformer_ctc/test_waves/
↪BAC009S0764W0122.wav', './tmp/icefall_asr_aishell_conformer_ctc/test_waves/
↪BAC009S0764W0123.wav']}
2021-11-18 07:37:38,684 INFO [pretrained.py:240] device: cuda:0
2021-11-18 07:37:38,684 INFO [pretrained.py:242] Creating model
2021-11-18 07:37:47,651 INFO [pretrained.py:259] Constructing Fbank computer
2021-11-18 07:37:47,654 INFO [pretrained.py:269] Reading sound files: ['./tmp/icefall_
↪asr_aishell_conformer_ctc/test_waves/BAC009S0764W0121.wav', './tmp/icefall_asr_aishell_
↪conformer_ctc/test_waves/BAC009S0764W0122.wav', './tmp/icefall_asr_aishell_conformer_
↪ctc/test_waves/BAC009S0764W0123.wav']
2021-11-18 07:37:47,659 INFO [pretrained.py:275] Decoding started
2021-11-18 07:37:47,752 INFO [pretrained.py:321] Loading HLG from ./tmp/icefall_asr_
```

(continues on next page)

(continued from previous page)

```
↪aishell_conformer_ctc/data/lang_char/HLG.pt
2021-11-18 07:37:51,887 INFO [pretrained.py:340] Use HLG decoding
2021-11-18 07:37:52,102 INFO [pretrained.py:370]
./tmp/icefall_asr_aishell_conformer_ctc/test_waves/BAC009S0764W0121.wav:


./tmp/icefall_asr_aishell_conformer_ctc/test_waves/BAC009S0764W0122.wav:


./tmp/icefall_asr_aishell_conformer_ctc/test_waves/BAC009S0764W0123.wav:



2021-11-18 07:37:52,102 INFO [pretrained.py:372] Decoding Done
```

HLG decoding + attention decoder rescoring

It extracts n paths from the lattice, recores the extracted paths with an attention decoder. The path with the highest score is the decoding result.

The command to run HLG decoding + attention decoder rescoring is:

```
$ cd egs/aishell/ASR
$ ./conformer_ctc/pretrained.py \
  --checkpoint ./tmp/icefall_asr_aishell_conformer_ctc/exp/pretrained.pt \
  --words-file ./tmp/icefall_asr_aishell_conformer_ctc/data/lang_char/words.txt \
  --HLG ./tmp/icefall_asr_aishell_conformer_ctc/data/lang_char/HLG.pt \
  --method attention-decoder \
  ./tmp/icefall_asr_aishell_conformer_ctc/test_waves/BAC009S0764W0121.wav \
  ./tmp/icefall_asr_aishell_conformer_ctc/test_waves/BAC009S0764W0122.wav \
  ./tmp/icefall_asr_aishell_conformer_ctc/test_waves/BAC009S0764W0123.wav
```

The output is below:

```
2021-11-18 07:42:05,965 INFO [pretrained.py:229] {'sample_rate': 16000, 'subsampling_
↪factor': 4, 'feature_dim': 80, 'nhead': 4, 'attention_dim': 512, 'num_decoder_layers':␣
↪6, 'vgg_frontend': False, 'use_feat_batchnorm': True, 'search_beam': 20, 'output_beam
↪': 8, 'min_active_states': 30, 'max_active_states': 10000, 'use_double_scores': True,
↪'env_info': {'k2-version': '1.9', 'k2-build-type': 'Release', 'k2-with-cuda': True,
↪'k2-git-sha1': 'f2fd997f752ed11bbef4c306652c433e83f9cf12', 'k2-git-date': 'Sun Sep 19␣
↪09:41:46 2021', 'lhotse-version': '0.11.0.dev+git.33cfe45.clean', 'torch-cuda-available
↪': True, 'torch-cuda-version': '10.1', 'python-version': '3.8', 'icefall-git-branch':
↪'aishell', 'icefall-git-sha1': 'd57a873-dirty', 'icefall-git-date': 'Wed Nov 17␣
↪19:53:25 2021', 'icefall-path': '/ceph-hw/kangwei/code/icefall_aishell3', 'k2-path': '/
↪ceph-hw/kangwei/code/k2_release/k2/k2/python/k2/__init__.py', 'lhotse-path': '/ceph-hw/
↪kangwei/code/lhotse/lhotse/__init__.py'}, 'checkpoint': './tmp/icefall_asr_aishell_
↪conformer_ctc/exp/pretrained.pt', 'tokens_file': None, 'words_file': './tmp/icefall_
↪asr_aishell_conformer_ctc/data/lang_char/words.txt', 'HLG': './tmp/icefall_asr_aishell_
↪conformer_ctc/data/lang_char/HLG.pt', 'method': 'attention-decoder', 'num_paths': 100,
↪'ngram_lm_scale': 0.3, 'attention_decoder_scale': 0.9, 'nbest_scale': 0.5, 'sos_id': 1,
↪ 'eos_id': 1, 'num_classes': 4336, 'sound_files': ['./tmp/icefall_asr_aishell_
```

(continues on next page)

(continued from previous page)

```
↪conformer_ctc/test_waves/BAC009S0764W0121.wav', './tmp/icefall_asr_aishell_conformer_
↪ctc/test_waves/BAC009S0764W0122.wav', './tmp/icefall_asr_aishell_conformer_ctc/test_
↪waves/BAC009S0764W0123.wav']}
2021-11-18 07:42:05,966 INFO [pretrained.py:240] device: cuda:0
2021-11-18 07:42:05,966 INFO [pretrained.py:242] Creating model
2021-11-18 07:42:16,821 INFO [pretrained.py:259] Constructing Fbank computer
2021-11-18 07:42:16,822 INFO [pretrained.py:269] Reading sound files: ['./tmp/icefall_
↪asr_aishell_conformer_ctc/test_waves/BAC009S0764W0121.wav', './tmp/icefall_asr_aishell_
↪conformer_ctc/test_waves/BAC009S0764W0122.wav', './tmp/icefall_asr_aishell_conformer_
↪ctc/test_waves/BAC009S0764W0123.wav']
2021-11-18 07:42:16,826 INFO [pretrained.py:275] Decoding started
2021-11-18 07:42:16,916 INFO [pretrained.py:321] Loading HLG from ./tmp/icefall_asr_
↪aishell_conformer_ctc/data/lang_char/HLG.pt
2021-11-18 07:42:21,115 INFO [pretrained.py:345] Use HLG + attention decoder rescoring
2021-11-18 07:42:21,888 INFO [pretrained.py:370]
./tmp/icefall_asr_aishell_conformer_ctc/test_waves/BAC009S0764W0121.wav:


./tmp/icefall_asr_aishell_conformer_ctc/test_waves/BAC009S0764W0122.wav:


./tmp/icefall_asr_aishell_conformer_ctc/test_waves/BAC009S0764W0123.wav:



2021-11-18 07:42:21,889 INFO [pretrained.py:372] Decoding Done
```

Colab notebook

We do provide a colab notebook for this recipe showing how to use a pre-trained model.

Hint: Due to limited memory provided by Colab, you have to upgrade to Colab Pro to run `HLG decoding + attention decoder rescoring`. Otherwise, you can only run `HLG decoding` with Colab.

Congratulations! You have finished the aishell ASR recipe with conformer CTC models in `icefall`.

If you want to deploy your trained model in C++, please read the following section.

Deployment with C++

This section describes how to deploy the pre-trained model in C++, without Python dependencies.

Hint: At present, it does NOT support streaming decoding.

First, let us compile k2 from source:

```
$ cd $HOME
$ git clone https://github.com/k2-fsa/k2
$ cd k2
$ git checkout v2.0-pre
```

Caution: You have to switch to the branch `v2.0-pre`!

```
$ mkdir build-release
$ cd build-release
$ cmake -DCMAKE_BUILD_TYPE=Release ..
$ make -j hlg_decode

# You will find four binaries in `./bin`, i.e. ./bin/hlg_decode,
```

Now you are ready to go!

Assume you have run:

```
$ cd k2/build-release
$ ln -s /path/to/icefall-asr-aishell-conformer-ctc ./
```

To view the usage of `./bin/hlg_decode`, run:

```
$ ./bin/hlg_decode
```

It will show you the following message:

```
Please provide --nn_model

This file implements decoding with an HLG decoding graph.

Usage:
  ./bin/hlg_decode \
    --use_gpu true \
    --nn_model <path to torch scripted pt file> \
    --hlg <path to HLG.pt> \
    --word_table <path to words.txt> \
    <path to foo.wav> \
    <path to bar.wav> \
    <more waves if any>

To see all possible options, use
  ./bin/hlg_decode --help

Caution:
 - Only sound files (*.wav) with single channel are supported.
 - It assumes the model is conformer_ctc/transformer.py from icefall.
   If you use a different model, you have to change the code
   related to `model.forward` in this file.
```

HLG decoding

```
./bin/hlg_decode \
  --use_gpu true \
  --nn_model icefall_asr_aishell_conformer_ctc/exp/cpu_jit.pt \
  --hlg icefall_asr_aishell_conformer_ctc/data/lang_char/HLG.pt \
  --word_table icefall_asr_aishell_conformer_ctc/data/lang_char/words.txt \
  icefall_asr_aishell_conformer_ctc/test_waves/BAC009S0764W0121.wav \
  icefall_asr_aishell_conformer_ctc/test_waves/BAC009S0764W0122.wav \
  icefall_asr_aishell_conformer_ctc/test_waves/BAC009S0764W0123.wav
```

The output is:

```
2021-11-18 14:48:20.89 [I] k2/torch/bin/hlg_decode.cu:115:int main(int, char**) Device:␣
↪cpu
2021-11-18 14:48:20.89 [I] k2/torch/bin/hlg_decode.cu:124:int main(int, char**) Load␣
↪wave files
2021-11-18 14:48:20.97 [I] k2/torch/bin/hlg_decode.cu:131:int main(int, char**) Build␣
↪Fbank computer
2021-11-18 14:48:20.98 [I] k2/torch/bin/hlg_decode.cu:142:int main(int, char**) Compute␣
↪features
2021-11-18 14:48:20.115 [I] k2/torch/bin/hlg_decode.cu:150:int main(int, char**) Load␣
↪neural network model
2021-11-18 14:48:20.693 [I] k2/torch/bin/hlg_decode.cu:165:int main(int, char**) Compute␣
↪nnet_output
2021-11-18 14:48:23.182 [I] k2/torch/bin/hlg_decode.cu:180:int main(int, char**) Load␣
↪icefall_asr_aishell_conformer_ctc/data/lang_char/HLG.pt
2021-11-18 14:48:33.489 [I] k2/torch/bin/hlg_decode.cu:185:int main(int, char**) Decoding
2021-11-18 14:48:45.217 [I] k2/torch/bin/hlg_decode.cu:216:int main(int, char**)
Decoding result:

icefall_asr_aishell_conformer_ctc/test_waves/BAC009S0764W0121.wav


icefall_asr_aishell_conformer_ctc/test_waves/BAC009S0764W0122.wav


icefall_asr_aishell_conformer_ctc/test_waves/BAC009S0764W0123.wav
```

There is a Colab notebook showing you how to run a torch scripted model in C++. Please see

Stateless Transducer

This tutorial shows you how to do transducer training in `icefall`.

Hint: Instead of using RNN-T or RNN transducer, we only use transducer here. As you will see, there are no RNNs in the model.

Hint: We assume you have read the page *Installation* and have setup the environment for `icefall`.

Hint: We recommend you to use a GPU or several GPUs to run this recipe.

In this tutorial, you will learn:

- (1) What does the transducer model look like
- (2) How to prepare data for training and decoding
- (3) How to start the training, either with a single GPU or with multiple GPUs
- (4) How to do decoding after training, with greedy search, beam search and, **modified beam search**
- (5) How to use a pre-trained model provided by us to transcribe sound files

The Model

The transducer model consists of 3 parts:

- **Encoder**: It is a conformer encoder with the following parameters
  - Number of heads: 8
  - Attention dim: 512
  - Number of layers: 12
  - Feedforward dim: 2048
- **Decoder**: We use a stateless model consisting of:
  - An embedding layer with embedding dim 512
  - A Conv1d layer with a default kernel size 2 (i.e. it sees 2 symbols of left-context by default)
- **Joiner**: It consists of a `nn.tanh()` and a `nn.Linear()`.

Caution: The decoder is stateless and very simple. It is borrowed from https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=9054419 (Rnn-Transducer with Stateless Prediction Network)

We make one modification to it: Place a Conv1d layer right after the embedding layer.

When using Chinese characters as modelling unit, whose vocabulary size is 4336 in this specific dataset, the number of parameters of the model is `87939824`, i.e., about `88 M`.

The Loss

We are using https://github.com/csukuangfj/optimized_transducer to compute the transducer loss, which removes extra paddings in loss computation to save memory.

Hint: `optimized_transducer` implements the technqiues proposed in Improving RNN Transducer Modeling for End-to-End Speech Recognition to save memory.

Furthermore, it supports `modified transducer`, limiting the maximum number of symbols that can be emitted per frame to 1, which simplifies the decoding process significantly. Also, the experiment results show that it does not degrade the performance.

See https://github.com/csukuangfj/optimized_transducer#modified-transducer for what exactly modified transducer is.

https://github.com/csukuangfj/transducer-loss-benchmarking shows that in the unpruned case `optimized_transducer` has the advantage about minimizing memory usage.

Todo: Add tutorial about `pruned_transducer_stateless` that uses k2 pruned transducer loss.

Hint: You can use:

```
pip install optimized_transducer
```

to install `optimized_transducer`. Refer to https://github.com/csukuangfj/optimized_transducer for other alternatives.

Data Preparation

To prepare the data for training, please use the following commands:

```
cd egs/aishell/ASR
./prepare.sh --stop-stage 4
./prepare.sh --stage 6 --stop-stage 6
```

Note: You can use `./prepare.sh`, though it will generate FSTs that are not used in transducer training.

When you finish running the script, you will get the following two folders:

- `data/fbank`: It saves the pre-computed features
- `data/lang_char`: It contains tokens that will be used in the training

Training

```
cd egs/aishell/ASR
./transducer_stateless_modified/train.py --help
```

shows you the training options that can be passed from the commandline. The following options are used quite often:

- `--exp-dir`

  The experiment folder to save logs and model checkpoints, defaults to `./transducer_stateless_modified/exp`.

- `--num-epochs`

  It is the number of epochs to train. For instance, `./transducer_stateless_modified/train.py --num-epochs 30` trains for 30 epochs and generates `epoch-0.pt`, `epoch-1.pt`, ..., `epoch-29.pt` in the folder set by `--exp-dir`.

- `--start-epoch`

  It's used to resume training. `./transducer_stateless_modified/train.py --start-epoch 10` loads the checkpoint from `exp_dir/epoch-9.pt` and starts training from epoch 10, based on the state from epoch 9.

- `--world-size`

  It is used for single-machine multi-GPU DDP training.

  - (a) If it is 1, then no DDP training is used.
  - (b) If it is 2, then GPU 0 and GPU 1 are used for DDP training.

  The following shows some use cases with it.

  Use case 1: You have 4 GPUs, but you only want to use GPU 0 and GPU 2 for training. You can do the following:

  ```
  $ cd egs/aishell/ASR
  $ export CUDA_VISIBLE_DEVICES="0,2"
  $ ./transducer_stateless_modified/train.py --world-size 2
  ```

  Use case 2: You have 4 GPUs and you want to use all of them for training. You can do the following:

  ```
  $ cd egs/aishell/ASR
  $ ./transducer_stateless_modified/train.py --world-size 4
  ```

  Use case 3: You have 4 GPUs but you only want to use GPU 3 for training. You can do the following:

  ```
  $ cd egs/aishell/ASR
  $ export CUDA_VISIBLE_DEVICES="3"
  $ ./transducer_stateless_modified/train.py --world-size 1
  ```

Caution: Only single-machine multi-GPU DDP training is implemented at present. There is an on-going PR https://github.com/k2-fsa/icefall/pull/63 that adds support for multi-machine multi-GPU DDP training.

- `--max-duration`

  It specifies the number of seconds over all utterances in a batch **before padding**. If you encounter CUDA OOM, please reduce it. For instance, if your are using V100 NVIDIA GPU with 32 GB RAM, we recommend you to set it to `300` when the vocabulary size is 500.

Hint: Due to padding, the number of seconds of all utterances in a batch will usually be larger than `--max-duration`.

A larger value for `--max-duration` may cause OOM during training, while a smaller value may increase the training time. You have to tune it.

- `--lr-factor`

  It controls the learning rate. If you use a single GPU for training, you may want to use a small value for it. If you use multiple GPUs for training, you may increase it.

- `--context-size`

  It specifies the kernel size in the decoder. The default value 2 means it functions as a tri-gram LM.

- `--modified-transducer-prob`

  It specifies the probability to use modified transducer loss. If it is 0, then no modified transducer is used; if it is 1, then it uses modified transducer loss for all batches. If it is `p`, it applies modified transducer with probability `p`.

There are some training options, e.g., number of warmup steps, that are not passed from the commandline. They are pre-configured by the function `get_params()` in transducer_stateless_modified/train.py

If you need to change them, please modify `./transducer_stateless_modified/train.py` directly.

Caution: The training set is perturbed by speed with two factors: 0.9 and 1.1. Each epoch actually processes `3x150 == 450` hours of data.

Training logs

Training logs and checkpoints are saved in the folder set by `--exp-dir` (defaults to `transducer_stateless_modified/exp`). You will find the following files in that directory:

- `epoch-0.pt`, `epoch-1.pt`, ...

  These are checkpoint files, containing model `state_dict` and optimizer `state_dict`. To resume training from some checkpoint, say `epoch-10.pt`, you can use:

  ```
  $ ./transducer_stateless_modified/train.py --start-epoch 11
  ```

- `tensorboard/`

  This folder contains TensorBoard logs. Training loss, validation loss, learning rate, etc, are recorded in these logs. You can visualize them by:

  ```
  $ cd transducer_stateless_modified/exp/tensorboard
  $ tensorboard dev upload --logdir . --name "Aishell transducer training␣
  ↪with icefall" --description "Training modified transducer, see https://
  ↪github.com/k2-fsa/icefall/pull/219"
  ```

  It will print something like below:

  ```
  TensorFlow installation not found - running with reduced feature set.
  Upload started and will continue reading any new data as it's added to the␣
  ↪logdir.

  To stop uploading, press Ctrl-C.

  New experiment created. View your TensorBoard at: https://tensorboard.dev/
  ↪experiment/laGZ6HrcQxOigbFD5E0Y3Q/

  [2022-03-03T14:29:45] Started scanning logdir.
  [2022-03-03T14:29:48] Total uploaded: 8477 scalars, 0 tensors, 0 binary␣
  ↪objects
  Listening for new data in logdir...
  ```

  Note there is a URL in the above output, click it and you will see the following screenshot:

- `log/log-train-xxxx`

  It is the detailed training log in text format, same as the one you saw printed to the console during training.

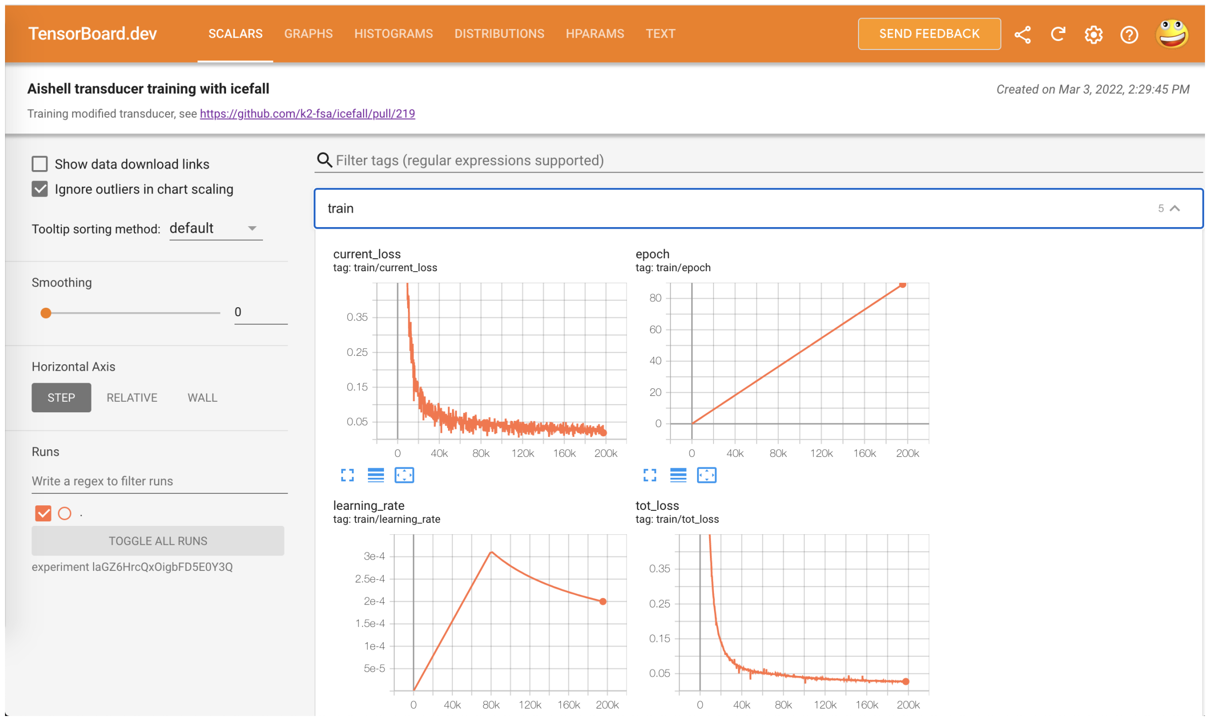

TensorBoard.dev
SCALARS
GRAPHS
HISTOGRAMS
DISTRIBUTIONS
HPARAMS
TEXT
SEND FEEDBACK
Aishell transducer training with icefall
Training modified transducer, see https://github.com/k2-fsa/icefall/pull/219
Created on Mar 3, 2022, 2:29:45 PM
Show data download links
Ignore outliers in chart scaling
Tooltip sorting method: default
Smoothing
0
Horizontal Axis
STEP
RELATIVE
WALL
Runs
Write a regex to filter runs
TOGGLE ALL RUNS
experiment laGZ6HrcQxOigbFD5E0Y3Q
Filter tags (regular expressions supported)
train
current_loss
tag: train/current_loss
epoch
tag: train/epoch
learning_rate
tag: train/learning_rate
tot_loss
tag: train/tot_loss

Fig. 6.3: TensorBoard screenshot.

Usage examples

The following shows typical use cases:

Case 1

```
$ cd egs/aishell/ASR
$ ./transducer_stateless_modified/train.py --max-duration 250
```

It uses `--max-duration` of 250 to avoid OOM.

Case 2

```
$ cd egs/aishell/ASR
$ export CUDA_VISIBLE_DEVICES="0,3"
$ ./transducer_stateless_modified/train.py --world-size 2
```

It uses GPU 0 and GPU 3 for DDP training.

Case 3

```
$ cd egs/aishell/ASR
$ ./transducer_stateless_modified/train.py --num-epochs 10 --start-epoch 3
```

It loads checkpoint `./transducer_stateless_modified/exp/epoch-2.pt` and starts training from epoch 3. Also, it trains for 10 epochs.

Decoding

The decoding part uses checkpoints saved by the training part, so you have to run the training part first.

```
$ cd egs/aishell/ASR
$ ./transducer_stateless_modified/decode.py --help
```

shows the options for decoding.

The commonly used options are:

- `--method`

  This specifies the decoding method. Currently, it supports:

  - **greedy_search**. You can provide the commandline option `--max-sym-per-frame` to limit the maximum number of symbols that can be emitted per frame.
  - **beam_search**. You can provide the commandline option `--beam-size`.
  - **modified_beam_search**. You can also provide the commandline option `--beam-size`. To use this method, we assume that you have trained your model with modified transducer, i.e., used the option `--modified-transducer-prob` in the training.

  The following command uses greedy search for decoding

  ```
  $ cd egs/aishell/ASR
  $ ./transducer_stateless_modified/decode.py \
          --epoch 64 \
          --avg 33 \
          --exp-dir ./transducer_stateless_modified/exp \
          --max-duration 100 \
          --decoding-method greedy_search \
          --max-sym-per-frame 1
  ```

  The following command uses beam search for decoding

  ```
  $ cd egs/aishell/ASR
  $ ./transducer_stateless_modified/decode.py \
          --epoch 64 \
          --avg 33 \
          --exp-dir ./transducer_stateless_modified/exp \
          --max-duration 100 \
          --decoding-method beam_search \
          --beam-size 4
  ```

  The following command uses `modified` beam search for decoding

```
$ cd egs/aishell/ASR
$ ./transducer_stateless_modified/decode.py \
        --epoch 64 \
        --avg 33 \
        --exp-dir ./transducer_stateless_modified/exp \
        --max-duration 100 \
        --decoding-method modified_beam_search \
        --beam-size 4
```

- `--max-duration`

  It has the same meaning as the one used in training. A larger value may cause OOM.

- `--epoch`

  It specifies the checkpoint from which epoch that should be used for decoding.

- `--avg`

  It specifies the number of models to average. For instance, if it is 3 and if `--epoch=10`, then it averages the checkpoints `epoch-8.pt`, `epoch-9.pt`, and `epoch-10.pt` and the averaged checkpoint is used for decoding.

After decoding, you can find the decoding logs and results in *exp_dir/log/<decoding_method>*, e.g., `exp_dir/log/greedy_search`.

Pre-trained Model

We have uploaded a pre-trained model to https://huggingface.co/csukuangfj/icefall-aishell-transducer-stateless-modified-2022-03-01

We describe how to use the pre-trained model to transcribe a sound file or multiple sound files in the following.

Install kaldifeat

kaldifeat is used to extract features for a single sound file or multiple sound files at the same time.

Please refer to https://github.com/csukuangfj/kaldifeat for installation.

Download the pre-trained model

The following commands describe how to download the pre-trained model:

```
$ cd egs/aishell/ASR
$ mkdir tmp
$ cd tmp
$ git lfs install
$ git clone https://huggingface.co/csukuangfj/icefall-aishell-transducer-stateless-
↪modified-2022-03-01
```

Caution: You have to use `git lfs` to download the pre-trained model.

After downloading, you will have the following files:

```
$ cd egs/aishell/ASR
$ tree tmp/icefall-aishell-transducer-stateless-modified-2022-03-01
```

```
tmp/icefall-aishell-transducer-stateless-modified-2022-03-01/
|-- README.md
|-- data
|   `-- lang_char
|       |-- L.pt
|       |-- lexicon.txt
|       |-- tokens.txt
|       `-- words.txt
|-- exp
|   `-- pretrained.pt
|-- log
|   |-- errs-test-beam_4-epoch-64-avg-33-beam-4.txt
|   |-- errs-test-greedy_search-epoch-64-avg-33-context-2-max-sym-per-frame-1.txt
|   |-- log-decode-epoch-64-avg-33-beam-4-2022-03-02-12-05-03
|   |-- log-decode-epoch-64-avg-33-context-2-max-sym-per-frame-1-2022-02-28-18-13-07
|   |-- recogs-test-beam_4-epoch-64-avg-33-beam-4.txt
|   `-- recogs-test-greedy_search-epoch-64-avg-33-context-2-max-sym-per-frame-1.txt
`-- test_wavs
    |-- BAC009S0764W0121.wav
    |-- BAC009S0764W0122.wav
    |-- BAC009S0764W0123.wav
    `-- transcript.txt

5 directories, 16 files
```

File descriptions:

- `data/lang_char`

  It contains language related files. You can find the vocabulary size in `tokens.txt`.

- `exp/pretrained.pt`

  It contains pre-trained model parameters, obtained by averaging checkpoints from `epoch-32.pt` to `epoch-64.pt`. Note: We have removed optimizer `state_dict` to reduce file size.

- `log`

  It contains decoding logs and decoded results.

- `test_wavs`

  It contains some test sound files from Aishell `test` dataset.

The information of the test sound files is listed below:

```
$ soxi tmp/icefall-aishell-transducer-stateless-modified-2022-03-01/test_wavs/*.wav

Input File     : 'tmp/icefall-aishell-transducer-stateless-modified-2022-03-01/test_wavs/
↪BAC009S0764W0121.wav'
Channels       : 1
Sample Rate    : 16000
Precision      : 16-bit
Duration       : 00:00:04.20 = 67263 samples ~ 315.295 CDDA sectors
```

(continues on next page)

(continued from previous page)

```
File Size      : 135k
Bit Rate       : 256k
Sample Encoding: 16-bit Signed Integer PCM


Input File     : 'tmp/icefall-aishell-transducer-stateless-modified-2022-03-01/test_wavs/
↪BAC009S0764W0122.wav'
Channels       : 1
Sample Rate    : 16000
Precision      : 16-bit
Duration       : 00:00:04.12 = 65840 samples ~ 308.625 CDDA sectors
File Size      : 132k
Bit Rate       : 256k
Sample Encoding: 16-bit Signed Integer PCM


Input File     : 'tmp/icefall-aishell-transducer-stateless-modified-2022-03-01/test_wavs/
↪BAC009S0764W0123.wav'
Channels       : 1
Sample Rate    : 16000
Precision      : 16-bit
Duration       : 00:00:04.00 = 64000 samples ~ 300 CDDA sectors
File Size      : 128k
Bit Rate       : 256k
Sample Encoding: 16-bit Signed Integer PCM

Total Duration of 3 files: 00:00:12.32
```

Usage

```
$ cd egs/aishell/ASR
$ ./transducer_stateless_modified/pretrained.py --help
```

displays the help information.

It supports three decoding methods:

- greedy search
- beam search
- modified beam search

Note: In modified beam search, it limits the maximum number of symbols that can be emitted per frame to 1. To use this method, you have to ensure that your model has been trained with the option `--modified-transducer-prob`. Otherwise, it may give you poor results.

Greedy search

The command to run greedy search is given below:

```
$ cd egs/aishell/ASR
$ ./transducer_stateless_modified/pretrained.py \
    --checkpoint ./tmp/icefall-aishell-transducer-stateless-modified-2022-03-01/exp/
↪pretrained.pt \
    --lang-dir ./tmp/icefall-aishell-transducer-stateless-modified-2022-03-01/data/lang_
↪char \
    --method greedy_search \
    ./tmp/icefall-aishell-transducer-stateless-modified-2022-03-01/test_wavs/
↪BAC009S0764W0121.wav \
    ./tmp/icefall-aishell-transducer-stateless-modified-2022-03-01/test_wavs/
↪BAC009S0764W0122.wav \
    ./tmp/icefall-aishell-transducer-stateless-modified-2022-03-01/test_wavs/
↪BAC009S0764W0123.wav
```

The output is as follows:

```
2022-03-03 15:35:26,531 INFO [pretrained.py:239] device: cuda:0
2022-03-03 15:35:26,994 INFO [lexicon.py:176] Loading pre-compiled tmp/icefall-aishell-
↪transducer-stateless-modified-2022-03-01/data/lang_char/Linv.pt
2022-03-03 15:35:27,027 INFO [pretrained.py:246] {'feature_dim': 80, 'encoder_out_dim':␣
↪512, 'subsampling_factor': 4, 'attention_dim': 512, 'nhead': 8, 'dim_feedforward':␣
↪2048, 'num_encoder_layers': 12, 'vgg_frontend': False, 'env_info': {'k2-version': '1.13
↪', 'k2-build-type': 'Release', 'k2-with-cuda': True, 'k2-git-sha1':
↪'f4fefe4882bc0ae59af951da3f47335d5495ef71', 'k2-git-date': 'Thu Feb 10 15:16:02 2022',
↪'lhotse-version': '1.0.0.dev+missing.version.file', 'torch-cuda-available': True,
↪'torch-cuda-version': '10.2', 'python-version': '3.8', 'icefall-git-branch': 'master',
↪'icefall-git-sha1': '50d2281-clean', 'icefall-git-date': 'Wed Mar 2 16:02:38 2022',
↪'icefall-path': '/ceph-fj/fangjun/open-source-2/icefall-aishell', 'k2-path': '/ceph-fj/
↪fangjun/open-source-2/k2-multi-datasets/k2/python/k2/__init__.py', 'lhotse-path': '/
↪ceph-fj/fangjun/open-source-2/lhotse-aishell/lhotse/__init__.py', 'hostname': 'de-
↪74279-k2-train-2-0815224919-75d558775b-mmnv8', 'IP address': '10.177.72.138'}, 'sample_
↪rate': 16000, 'checkpoint': './tmp/icefall-aishell-transducer-stateless-modified-2022-
↪03-01/exp/pretrained.pt', 'lang_dir': PosixPath('tmp/icefall-aishell-transducer-
↪stateless-modified-2022-03-01/data/lang_char'), 'method': 'greedy_search', 'sound_files
↪': ['./tmp/icefall-aishell-transducer-stateless-modified-2022-03-01/test_wavs/
↪BAC009S0764W0121.wav', './tmp/icefall-aishell-transducer-stateless-modified-2022-03-01/
↪test_wavs/BAC009S0764W0122.wav', './tmp/icefall-aishell-transducer-stateless-modified-
↪2022-03-01/test_wavs/BAC009S0764W0123.wav'], 'beam_size': 4, 'context_size': 2, 'max_
↪sym_per_frame': 3, 'blank_id': 0, 'vocab_size': 4336}
2022-03-03 15:35:27,027 INFO [pretrained.py:248] About to create model
2022-03-03 15:35:36,878 INFO [pretrained.py:257] Constructing Fbank computer
2022-03-03 15:35:36,880 INFO [pretrained.py:267] Reading sound files: ['./tmp/icefall-
↪aishell-transducer-stateless-modified-2022-03-01/test_wavs/BAC009S0764W0121.wav', './
↪tmp/icefall-aishell-transducer-stateless-modified-2022-03-01/test_wavs/
↪BAC009S0764W0122.wav', './tmp/icefall-aishell-transducer-stateless-modified-2022-03-01/
↪test_wavs/BAC009S0764W0123.wav']
2022-03-03 15:35:36,891 INFO [pretrained.py:273] Decoding started
/ceph-fj/fangjun/open-source-2/icefall-aishell/egs/aishell/ASR/transducer_stateless_
↪modified/conformer.py:113: UserWarning: __floordiv__ is deprecated, and its behavior␣
```

(continues on next page)

(continued from previous page)

```
↪will change in a future version of pytorch. It currently rounds toward 0 (like the
↪'trunc' function NOT 'floor'). This results in incorrect rounding for negative values.␣
↪To keep the current behavior, use torch.div(a, b, rounding_mode='trunc'), or for␣
↪actual floor division, use torch.div(a, b, rounding_mode='floor').
  lengths = ((x_lens - 1) // 2 - 1) // 2
2022-03-03 15:35:37,163 INFO [pretrained.py:320]
./tmp/icefall-aishell-transducer-stateless-modified-2022-03-01/test_wavs/
↪BAC009S0764W0121.wav:


./tmp/icefall-aishell-transducer-stateless-modified-2022-03-01/test_wavs/
↪BAC009S0764W0122.wav:


./tmp/icefall-aishell-transducer-stateless-modified-2022-03-01/test_wavs/
↪BAC009S0764W0123.wav:


2022-03-03 15:35:37,163 INFO [pretrained.py:322] Decoding Done
```

Beam search

The command to run beam search is given below:

```
$ cd egs/aishell/ASR

$ ./transducer_stateless_modified/pretrained.py \
    --checkpoint ./tmp/icefall-aishell-transducer-stateless-modified-2022-03-01/exp/
↪pretrained.pt \
    --lang-dir ./tmp/icefall-aishell-transducer-stateless-modified-2022-03-01/data/lang_
↪char \
    --method beam_search \
    --beam-size 4 \
    ./tmp/icefall-aishell-transducer-stateless-modified-2022-03-01/test_wavs/
↪BAC009S0764W0121.wav \
    ./tmp/icefall-aishell-transducer-stateless-modified-2022-03-01/test_wavs/
↪BAC009S0764W0122.wav \
    ./tmp/icefall-aishell-transducer-stateless-modified-2022-03-01/test_wavs/
↪BAC009S0764W0123.wav
```

The output is as follows:

```
2022-03-03 15:39:09,285 INFO [pretrained.py:239] device: cuda:0
2022-03-03 15:39:09,708 INFO [lexicon.py:176] Loading pre-compiled tmp/icefall-aishell-
↪transducer-stateless-modified-2022-03-01/data/lang_char/Linv.pt
2022-03-03 15:39:09,759 INFO [pretrained.py:246] {'feature_dim': 80, 'encoder_out_dim':␣
↪512, 'subsampling_factor': 4, 'attention_dim': 512, 'nhead': 8, 'dim_feedforward':␣
↪2048, 'num_encoder_layers': 12, 'vgg_frontend': False, 'env_info': {'k2-version': '1.13
↪', 'k2-build-type': 'Release', 'k2-with-cuda': True, 'k2-git-sha1':
↪'f4fefe4882bc0ae59af951da3f47335d5495ef71', 'k2-git-date': 'Thu Feb 10 15:16:02 2022',
↪'lhotse-version': '1.0.0.dev+missing.version.file', 'torch-cuda-available': True,
```

(continues on next page)

(continued from previous page)

```
↪'torch-cuda-version': '10.2', 'python-version': '3.8', 'icefall-git-branch': 'master',
↪'icefall-git-sha1': '50d2281-clean', 'icefall-git-date': 'Wed Mar 2 16:02:38 2022',
↪'icefall-path': '/ceph-fj/fangjun/open-source-2/icefall-aishell', 'k2-path': '/ceph-fj/
↪fangjun/open-source-2/k2-multi-datasets/k2/python/k2/__init__.py', 'lhotse-path': '/
↪ceph-fj/fangjun/open-source-2/lhotse-aishell/lhotse/__init__.py', 'hostname': 'de-
↪74279-k2-train-2-0815224919-75d558775b-mmnv8', 'IP address': '10.177.72.138'}, 'sample_
↪rate': 16000, 'checkpoint': './tmp/icefall-aishell-transducer-stateless-modified-2022-
↪03-01/exp/pretrained.pt', 'lang_dir': PosixPath('tmp/icefall-aishell-transducer-
↪stateless-modified-2022-03-01/data/lang_char'), 'method': 'beam_search', 'sound_files
↪': ['./tmp/icefall-aishell-transducer-stateless-modified-2022-03-01/test_wavs/
↪BAC009S0764W0121.wav', './tmp/icefall-aishell-transducer-stateless-modified-2022-03-01/
↪test_wavs/BAC009S0764W0122.wav', './tmp/icefall-aishell-transducer-stateless-modified-
↪2022-03-01/test_wavs/BAC009S0764W0123.wav'], 'beam_size': 4, 'context_size': 2, 'max_
↪sym_per_frame': 3, 'blank_id': 0, 'vocab_size': 4336}
2022-03-03 15:39:09,760 INFO [pretrained.py:248] About to create model
2022-03-03 15:39:18,919 INFO [pretrained.py:257] Constructing Fbank computer
2022-03-03 15:39:18,922 INFO [pretrained.py:267] Reading sound files: ['./tmp/icefall-
↪aishell-transducer-stateless-modified-2022-03-01/test_wavs/BAC009S0764W0121.wav', './
↪tmp/icefall-aishell-transducer-stateless-modified-2022-03-01/test_wavs/
↪BAC009S0764W0122.wav', './tmp/icefall-aishell-transducer-stateless-modified-2022-03-01/
↪test_wavs/BAC009S0764W0123.wav']
2022-03-03 15:39:18,929 INFO [pretrained.py:273] Decoding started
/ceph-fj/fangjun/open-source-2/icefall-aishell/egs/aishell/ASR/transducer_stateless_
↪modified/conformer.py:113: UserWarning: __floordiv__ is deprecated, and its behavior␣
↪will change in a future version of pytorch. It currently rounds toward 0 (like the
↪'trunc' function NOT 'floor'). This results in incorrect rounding for negative values.␣
↪To keep the current behavior, use torch.div(a, b, rounding_mode='trunc'), or for␣
↪actual floor division, use torch.div(a, b, rounding_mode='floor').
  lengths = ((x_lens - 1) // 2 - 1) // 2
2022-03-03 15:39:21,046 INFO [pretrained.py:320]
./tmp/icefall-aishell-transducer-stateless-modified-2022-03-01/test_wavs/
↪BAC009S0764W0121.wav:


./tmp/icefall-aishell-transducer-stateless-modified-2022-03-01/test_wavs/
↪BAC009S0764W0122.wav:


./tmp/icefall-aishell-transducer-stateless-modified-2022-03-01/test_wavs/
↪BAC009S0764W0123.wav:


2022-03-03 15:39:21,047 INFO [pretrained.py:322] Decoding Done
```

Modified Beam search

The command to run modified beam search is given below:

```
$ cd egs/aishell/ASR

$ ./transducer_stateless_modified/pretrained.py \
    --checkpoint ./tmp/icefall-aishell-transducer-stateless-modified-2022-03-01/exp/
↪pretrained.pt \
    --lang-dir ./tmp/icefall-aishell-transducer-stateless-modified-2022-03-01/data/lang_
↪char \
    --method modified_beam_search \
    --beam-size 4 \
    ./tmp/icefall-aishell-transducer-stateless-modified-2022-03-01/test_wavs/
↪BAC009S0764W0121.wav \
    ./tmp/icefall-aishell-transducer-stateless-modified-2022-03-01/test_wavs/
↪BAC009S0764W0122.wav \
    ./tmp/icefall-aishell-transducer-stateless-modified-2022-03-01/test_wavs/
↪BAC009S0764W0123.wav
```

The output is as follows:

```
2022-03-03 15:41:23,319 INFO [pretrained.py:239] device: cuda:0
2022-03-03 15:41:23,798 INFO [lexicon.py:176] Loading pre-compiled tmp/icefall-aishell-
↪transducer-stateless-modified-2022-03-01/data/lang_char/Linv.pt
2022-03-03 15:41:23,831 INFO [pretrained.py:246] {'feature_dim': 80, 'encoder_out_dim':␣
↪512, 'subsampling_factor': 4, 'attention_dim': 512, 'nhead': 8, 'dim_feedforward':␣
↪2048, 'num_encoder_layers': 12, 'vgg_frontend': False, 'env_info': {'k2-version': '1.13
↪', 'k2-build-type': 'Release', 'k2-with-cuda': True, 'k2-git-sha1':
↪'f4fefe4882bc0ae59af951da3f47335d5495ef71', 'k2-git-date': 'Thu Feb 10 15:16:02 2022',
↪'lhotse-version': '1.0.0.dev+missing.version.file', 'torch-cuda-available': True,
↪'torch-cuda-version': '10.2', 'python-version': '3.8', 'icefall-git-branch': 'master',
↪'icefall-git-sha1': '50d2281-clean', 'icefall-git-date': 'Wed Mar 2 16:02:38 2022',
↪'icefall-path': '/ceph-fj/fangjun/open-source-2/icefall-aishell', 'k2-path': '/ceph-fj/
↪fangjun/open-source-2/k2-multi-datasets/k2/python/k2/__init__.py', 'lhotse-path': '/
↪ceph-fj/fangjun/open-source-2/lhotse-aishell/lhotse/__init__.py', 'hostname': 'de-
↪74279-k2-train-2-0815224919-75d558775b-mmnv8', 'IP address': '10.177.72.138'}, 'sample_
↪rate': 16000, 'checkpoint': './tmp/icefall-aishell-transducer-stateless-modified-2022-
↪03-01/exp/pretrained.pt', 'lang_dir': PosixPath('tmp/icefall-aishell-transducer-
↪stateless-modified-2022-03-01/data/lang_char'), 'method': 'modified_beam_search',
↪'sound_files': ['./tmp/icefall-aishell-transducer-stateless-modified-2022-03-01/test_
↪wavs/BAC009S0764W0121.wav', './tmp/icefall-aishell-transducer-stateless-modified-2022-
↪03-01/test_wavs/BAC009S0764W0122.wav', './tmp/icefall-aishell-transducer-stateless-
↪modified-2022-03-01/test_wavs/BAC009S0764W0123.wav'], 'beam_size': 4, 'context_size':␣
↪2, 'max_sym_per_frame': 3, 'blank_id': 0, 'vocab_size': 4336}
2022-03-03 15:41:23,831 INFO [pretrained.py:248] About to create model
2022-03-03 15:41:32,214 INFO [pretrained.py:257] Constructing Fbank computer
2022-03-03 15:41:32,215 INFO [pretrained.py:267] Reading sound files: ['./tmp/icefall-
↪aishell-transducer-stateless-modified-2022-03-01/test_wavs/BAC009S0764W0121.wav', './
↪tmp/icefall-aishell-transducer-stateless-modified-2022-03-01/test_wavs/
↪BAC009S0764W0122.wav', './tmp/icefall-aishell-transducer-stateless-modified-2022-03-01/
↪test_wavs/BAC009S0764W0123.wav']
2022-03-03 15:41:32,220 INFO [pretrained.py:273] Decoding started
```

(continues on next page)

(continued from previous page)

```
/ceph-fj/fangjun/open-source-2/icefall-aishell/egs/aishell/ASR/transducer_stateless_
↪modified/conformer.py:113: UserWarning: __floordiv__ is deprecated, and its behavior␣
↪will change in a future version of pytorch. It currently rounds toward 0 (like the
↪'trunc' function NOT 'floor'). This results in incorrect rounding for negative values.␣
↪To keep the current behavior, use torch.div(a, b, rounding_mode='trunc'), or for␣
↪actual floor division, use torch.div(a, b, rounding_mode='floor').
  lengths = ((x_lens - 1) // 2 - 1) // 2
/ceph-fj/fangjun/open-source-2/icefall-aishell/egs/aishell/ASR/transducer_stateless_
↪modified/beam_search.py:402: UserWarning: __floordiv__ is deprecated, and its behavior␣
↪will change in a future version of pytorch. It currently rounds toward 0 (like the
↪'trunc' function NOT 'floor'). This results in incorrect rounding for negative values.␣
↪To keep the current behavior, use torch.div(a, b, rounding_mode='trunc'), or for␣
↪actual floor division, use torch.div(a, b, rounding_mode='floor').
  topk_hyp_indexes = topk_indexes // logits.size(-1)
2022-03-03 15:41:32,583 INFO [pretrained.py:320]
./tmp/icefall-aishell-transducer-stateless-modified-2022-03-01/test_wavs/
↪BAC009S0764W0121.wav:


./tmp/icefall-aishell-transducer-stateless-modified-2022-03-01/test_wavs/
↪BAC009S0764W0122.wav:


./tmp/icefall-aishell-transducer-stateless-modified-2022-03-01/test_wavs/
↪BAC009S0764W0123.wav:


2022-03-03 15:41:32,583 INFO [pretrained.py:322] Decoding Done
```

Colab notebook

We provide a colab notebook for this recipe showing how to use a pre-trained model to transcribe sound files.

6.1.2 LibriSpeech

TDNN-LSTM-CTC

This tutorial shows you how to run a TDNN-LSTM-CTC model with the LibriSpeech dataset.

Hint: We assume you have read the page *Installation* and have setup the environment for `icefall`.

Data preparation

```
$ cd egs/librispeech/ASR
$ ./prepare.sh
```

The script `./prepare.sh` handles the data preparation for you, **automagically**. All you need to do is to run it.

The data preparation contains several stages, you can use the following two options:

- `--stage`
- `--stop-stage`

to control which stage(s) should be run. By default, all stages are executed.

For example,

```
$ cd egs/librispeech/ASR
$ ./prepare.sh --stage 0 --stop-stage 0
```

means to run only stage 0.

To run stage 2 to stage 5, use:

```
$ ./prepare.sh --stage 2 --stop-stage 5
```

We provide the following YouTube video showing how to run `./prepare.sh`.

Note: To get the latest news of next-gen Kaldi, please subscribe the following YouTube channel by Nadira Povey:

https://www.youtube.com/channel/UC_VaumpkmINz1pNkFXAN9mw

https://youtu.be/ofEIoJL-mGM

Training

Now describing the training of TDNN-LSTM-CTC model, contained in the tdnn_lstm_ctc folder.

The command to run the training part is:

```
$ cd egs/librispeech/ASR
$ export CUDA_VISIBLE_DEVICES="0,1,2,3"
$ ./tdnn_lstm_ctc/train.py --world-size 4
```

By default, it will run `20` epochs. Training logs and checkpoints are saved in `tdnn_lstm_ctc/exp`.

In `tdnn_lstm_ctc/exp`, you will find the following files:

- `epoch-0.pt`, `epoch-1.pt`, . . . , `epoch-19.pt`

  These are checkpoint files, containing model `state_dict` and optimizer `state_dict`. To resume training from some checkpoint, say `epoch-10.pt`, you can use:

  ```
  $ ./tdnn_lstm_ctc/train.py --start-epoch 11
  ```

- `tensorboard/`

  This folder contains TensorBoard logs. Training loss, validation loss, learning rate, etc, are recorded in these logs. You can visualize them by:

  ```
  $ cd tdnn_lstm_ctc/exp/tensorboard
  $ tensorboard dev upload --logdir . --description "TDNN LSTM training for␣
  ↪librispeech with icefall"
  ```

- `log/log-train-xxxx`

  It is the detailed training log in text format, same as the one you saw printed to the console during training.

To see available training options, you can use:

```
$ ./tdnn_lstm_ctc/train.py --help
```

Other training options, e.g., learning rate, results dir, etc., are pre-configured in the function `get_params()` in tdnn_lstm_ctc/train.py. Normally, you don't need to change them. You can change them by modifying the code, if you want.

Decoding

The decoding part uses checkpoints saved by the training part, so you have to run the training part first.

The command for decoding is:

```
$ export CUDA_VISIBLE_DEVICES="0"
$ ./tdnn_lstm_ctc/decode.py
```

You will see the WER in the output log.

Decoded results are saved in `tdnn_lstm_ctc/exp`.

```
$ ./tdnn_lstm_ctc/decode.py --help
```

shows you the available decoding options.

Some commonly used options are:

- `--epoch`

  You can select which checkpoint to be used for decoding. For instance, `./tdnn_lstm_ctc/decode.py --epoch 10` means to use `./tdnn_lstm_ctc/exp/epoch-10.pt` for decoding.

- `--avg`

  It's related to model averaging. It specifies number of checkpoints to be averaged. The averaged model is used for decoding. For example, the following command:

  ```
  $ ./tdnn_lstm_ctc/decode.py --epoch 10 --avg 3
  ```

  uses the average of `epoch-8.pt`, `epoch-9.pt` and `epoch-10.pt` for decoding.

- `--export`

  If it is `True`, i.e., `./tdnn_lstm_ctc/decode.py --export 1`, the code will save the averaged model to `tdnn_lstm_ctc/exp/pretrained.pt`. See *Pre-trained Model* for how to use it.

Pre-trained Model

We have uploaded the pre-trained model to https://huggingface.co/pkufool/icefall_asr_librispeech_tdnn-lstm_ctc.

The following shows you how to use the pre-trained model.

Install kaldifeat

kaldifeat is used to extract features for a single sound file or multiple sound files at the same time.

Please refer to https://github.com/csukuangfj/kaldifeat for installation.

Download the pre-trained model

```
$ cd egs/librispeech/ASR
$ mkdir tmp
$ cd tmp
$ git lfs install
$ git clone https://huggingface.co/pkufool/icefall_asr_librispeech_tdnn-lstm_ctc
```

Caution: You have to use `git lfs` to download the pre-trained model.

Caution: In order to use this pre-trained model, your k2 version has to be v1.7 or later.

After downloading, you will have the following files:

```
$ cd egs/librispeech/ASR
$ tree tmp
```

```
tmp/
`-- icefall_asr_librispeech_tdnn-lstm_ctc
    |-- README.md
    |-- data
    |   |-- lang_phone
    |   |   |-- HLG.pt
    |   |   |-- tokens.txt
    |   |   `-- words.txt
    |   `-- lm
    |       `-- G_4_gram.pt
    |-- exp
    |   `-- pretrained.pt
    `-- test_wavs
        |-- 1089-134686-0001.flac
        |-- 1221-135766-0001.flac
        |-- 1221-135766-0002.flac
        `-- trans.txt

6 directories, 10 files
```

File descriptions:

- `data/lang_phone/HLG.pt`

  It is the decoding graph.

- `data/lang_phone/tokens.txt`

  It contains tokens and their IDs.

- `data/lang_phone/words.txt`

  It contains words and their IDs.

- `data/lm/G_4_gram.pt`

  It is a 4-gram LM, useful for LM rescoring.

- `exp/pretrained.pt`

  It contains pre-trained model parameters, obtained by averaging checkpoints from `epoch-14.pt` to `epoch-19.pt`. Note: We have removed optimizer `state_dict` to reduce file size.

- `test_waves/*.flac`

  It contains some test sound files from LibriSpeech `test-clean` dataset.

- `test_waves/trans.txt`

  It contains the reference transcripts for the sound files in `test_waves/`.

The information of the test sound files is listed below:

```
$ soxi tmp/icefall_asr_librispeech_tdnn-lstm_ctc/test_wavs/*.flac

Input File     : 'tmp/icefall_asr_librispeech_tdnn-lstm_ctc/test_wavs/1089-134686-0001.
↪flac'
Channels       : 1
Sample Rate    : 16000
Precision      : 16-bit
Duration       : 00:00:06.62 = 106000 samples ~ 496.875 CDDA sectors
File Size      : 116k
Bit Rate       : 140k
Sample Encoding: 16-bit FLAC


Input File     : 'tmp/icefall_asr_librispeech_tdnn-lstm_ctc/test_wavs/1221-135766-0001.
↪flac'
Channels       : 1
Sample Rate    : 16000
Precision      : 16-bit
Duration       : 00:00:16.71 = 267440 samples ~ 1253.62 CDDA sectors
File Size      : 343k
Bit Rate       : 164k
Sample Encoding: 16-bit FLAC


Input File     : 'tmp/icefall_asr_librispeech_tdnn-lstm_ctc/test_wavs/1221-135766-0002.
↪flac'
Channels       : 1
Sample Rate    : 16000
```

(continues on next page)

(continued from previous page)

```
Precision      : 16-bit
Duration       : 00:00:04.83 = 77200 samples ~ 361.875 CDDA sectors
File Size      : 105k
Bit Rate       : 174k
Sample Encoding: 16-bit FLAC

Total Duration of 3 files: 00:00:28.16
```

Inference with a pre-trained model

```
$ cd egs/librispeech/ASR
$ ./tdnn_lstm_ctc/pretrained.py --help
```

shows the usage information of `./tdnn_lstm_ctc/pretrained.py`.

To decode with `1best` method, we can use:

```
./tdnn_lstm_ctc/pretrained.py \
  --checkpoint ./tmp/icefall_asr_librispeech_tdnn-lstm_ctc/exp/pretraind.pt \
  --words-file ./tmp/icefall_asr_librispeech_tdnn-lstm_ctc/data/lang_phone/words.txt \
  --HLG ./tmp/icefall_asr_librispeech_tdnn-lstm_ctc/data/lang_phone/HLG.pt \
  ./tmp/icefall_asr_librispeech_tdnn-lstm_ctc/test_wavs/1089-134686-0001.flac \
  ./tmp/icefall_asr_librispeech_tdnn-lstm_ctc/test_wavs/1221-135766-0001.flac \
  ./tmp/icefall_asr_librispeech_tdnn-lstm_ctc/test_wavs/1221-135766-0002.flac
```

The output is:

```
2021-08-24 16:57:13,315 INFO [pretrained.py:168] device: cuda:0
2021-08-24 16:57:13,315 INFO [pretrained.py:170] Creating model
2021-08-24 16:57:18,331 INFO [pretrained.py:182] Loading HLG from ./tmp/icefall_asr_
↪librispeech_tdnn-lstm_ctc/data/lang_phone/HLG.pt
2021-08-24 16:57:27,581 INFO [pretrained.py:199] Constructing Fbank computer
2021-08-24 16:57:27,584 INFO [pretrained.py:209] Reading sound files: ['./tmp/icefall_
↪asr_librispeech_tdnn-lstm_ctc/test_wavs/1089-134686-0001.flac', './tmp/icefall_asr_
↪librispeech_tdnn-lstm_ctc/test_wavs/1221-135766-0001.flac', './tmp/icefall_asr_
↪librispeech_tdnn-lstm_ctc/test_wavs/1221-135766-0002.flac']
2021-08-24 16:57:27,599 INFO [pretrained.py:215] Decoding started
2021-08-24 16:57:27,791 INFO [pretrained.py:245] Use HLG decoding
2021-08-24 16:57:28,098 INFO [pretrained.py:266]
./tmp/icefall_asr_librispeech_tdnn-lstm_ctc/test_wavs/1089-134686-0001.flac:
AFTER EARLY NIGHTFALL THE YELLOW LAMPS WOULD LIGHT UP HERE AND THERE THE SQUALID QUARTER␣
↪OF THE BROTHELS

./tmp/icefall_asr_librispeech_tdnn-lstm_ctc/test_wavs/1221-135766-0001.flac:
GOD AS A DIRECT CONSEQUENCE OF THE SIN WHICH MAN THUS PUNISHED HAD GIVEN HER A LOVELY␣
↪CHILD WHOSE PLACE WAS ON THAT SAME DISHONORED BOSOM TO CONNECT HER PARENT FOREVER WITH␣
↪THE RACE AND DESCENT OF MORTALS AND TO BE FINALLY A BLESSED SOUL IN HEAVEN

./tmp/icefall_asr_librispeech_tdnn-lstm_ctc/test_wavs/1221-135766-0002.flac:
YET THESE THOUGHTS AFFECTED HESTER PRYNNE LESS WITH HOPE THAN APPREHENSION

```

(continues on next page)

(continued from previous page)

```
2021-08-24 16:57:28,099 INFO [pretrained.py:268] Decoding Done
```

To decode with `whole-lattice-rescoring` methond, you can use

```
./tdnn_lstm_ctc/pretrained.py \
  --checkpoint ./tmp/icefall_asr_librispeech_tdnn-lstm_ctc/exp/pretraind.pt \
  --words-file ./tmp/icefall_asr_librispeech_tdnn-lstm_ctc/data/lang_phone/words.txt \
  --HLG ./tmp/icefall_asr_librispeech_tdnn-lstm_ctc/data/lang_phone/HLG.pt \
  --method whole-lattice-rescoring \
  --G ./tmp/icefall_asr_librispeech_tdnn-lstm_ctc/data/lm/G_4_gram.pt \
  --ngram-lm-scale 0.8 \
  ./tmp/icefall_asr_librispeech_tdnn-lstm_ctc/test_wavs/1089-134686-0001.flac \
  ./tmp/icefall_asr_librispeech_tdnn-lstm_ctc/test_wavs/1221-135766-0001.flac \
  ./tmp/icefall_asr_librispeech_tdnn-lstm_ctc/test_wavs/1221-135766-0002.flac
```

The decoding output is:

```
2021-08-24 16:39:24,725 INFO [pretrained.py:168] device: cuda:0
2021-08-24 16:39:24,725 INFO [pretrained.py:170] Creating model
2021-08-24 16:39:29,403 INFO [pretrained.py:182] Loading HLG from ./tmp/icefall_asr_
↪librispeech_tdnn-lstm_ctc/data/lang_phone/HLG.pt
2021-08-24 16:39:40,631 INFO [pretrained.py:190] Loading G from ./tmp/icefall_asr_
↪librispeech_tdnn-lstm_ctc/data/lm/G_4_gram.pt
2021-08-24 16:39:53,098 INFO [pretrained.py:199] Constructing Fbank computer
2021-08-24 16:39:53,107 INFO [pretrained.py:209] Reading sound files: ['./tmp/icefall_
↪asr_librispeech_tdnn-lstm_ctc/test_wavs/1089-134686-0001.flac', './tmp/icefall_asr_
↪librispeech_tdnn-lstm_ctc/test_wavs/1221-135766-0001.flac', './tmp/icefall_asr_
↪librispeech_tdnn-lstm_ctc/test_wavs/1221-135766-0002.flac']
2021-08-24 16:39:53,121 INFO [pretrained.py:215] Decoding started
2021-08-24 16:39:53,443 INFO [pretrained.py:250] Use HLG decoding + LM rescoring
2021-08-24 16:39:54,010 INFO [pretrained.py:266]
./tmp/icefall_asr_librispeech_tdnn-lstm_ctc/test_wavs/1089-134686-0001.flac:
AFTER EARLY NIGHTFALL THE YELLOW LAMPS WOULD LIGHT UP HERE AND THERE THE SQUALID QUARTER␣
↪OF THE BROTHELS

./tmp/icefall_asr_librispeech_tdnn-lstm_ctc/test_wavs/1221-135766-0001.flac:
GOD AS A DIRECT CONSEQUENCE OF THE SIN WHICH MAN THUS PUNISHED HAD GIVEN HER A LOVELY␣
↪CHILD WHOSE PLACE WAS ON THAT SAME DISHONORED BOSOM TO CONNECT HER PARENT FOREVER WITH␣
↪THE RACE AND DESCENT OF MORTALS AND TO BE FINALLY A BLESSED SOUL IN HEAVEN

./tmp/icefall_asr_librispeech_tdnn-lstm_ctc/test_wavs/1221-135766-0002.flac:
YET THESE THOUGHTS AFFECTED HESTER PRYNNE LESS WITH HOPE THAN APPREHENSION


2021-08-24 16:39:54,010 INFO [pretrained.py:268] Decoding Done
```

Colab notebook

We provide a colab notebook for decoding with pre-trained model.

Congratulations! You have finished the TDNN-LSTM-CTC recipe on librispeech in `icefall`.

Conformer CTC

This tutorial shows you how to run a conformer ctc model with the LibriSpeech dataset.

Hint: We assume you have read the page *Installation* and have setup the environment for `icefall`.

Hint: We recommend you to use a GPU or several GPUs to run this recipe.

In this tutorial, you will learn:

- (1) How to prepare data for training and decoding
- (2) How to start the training, either with a single GPU or multiple GPUs
- (3) How to do decoding after training, with n-gram LM rescoring and attention decoder rescoring
- (4) How to use a pre-trained model, provided by us
- (5) How to deploy your trained model in C++, without Python dependencies

Data preparation

```
$ cd egs/librispeech/ASR
$ ./prepare.sh
```

The script `./prepare.sh` handles the data preparation for you, **automagically**. All you need to do is to run it.

The data preparation contains several stages, you can use the following two options:

- `--stage`
- `--stop-stage`

to control which stage(s) should be run. By default, all stages are executed.

For example,

```
$ cd egs/librispeech/ASR
$ ./prepare.sh --stage 0 --stop-stage 0
```

means to run only stage 0.

To run stage 2 to stage 5, use:

```
$ ./prepare.sh --stage 2 --stop-stage 5
```

Hint: If you have pre-downloaded the LibriSpeech dataset and the musan dataset, say, they are saved in `/tmp/LibriSpeech` and `/tmp/musan`, you can modify the `dl_dir` variable in `./prepare.sh` to point to `/tmp` so that `./prepare.sh` won't re-download them.

Note: All generated files by `./prepare.sh`, e.g., features, lexicon, etc, are saved in `./data` directory.

We provide the following YouTube video showing how to run `./prepare.sh`.

Note: To get the latest news of next-gen Kaldi, please subscribe the following YouTube channel by Nadira Povey:

https://www.youtube.com/channel/UC_VaumpkmINz1pNkFXAN9mw

https://youtu.be/ofEIoJL-mGM

Training

Configurable options

```
$ cd egs/librispeech/ASR
$ ./conformer_ctc/train.py --help
```

shows you the training options that can be passed from the commandline. The following options are used quite often:

- `--full-libri`

  If it's True, the training part uses all the training data, i.e., 960 hours. Otherwise, the training part uses only the subset `train-clean-100`, which has 100 hours of training data.

  Caution: The training set is perturbed by speed with two factors: 0.9 and 1.1. If `--full-libri` is True, each epoch actually processes `3x960 == 2880` hours of data.

- `--num-epochs`

  It is the number of epochs to train. For instance, `./conformer_ctc/train.py --num-epochs 30` trains for 30 epochs and generates `epoch-0.pt`, `epoch-1.pt`, . . . , `epoch-29.pt` in the folder `./conformer_ctc/exp`.

- `--start-epoch`

  It's used to resume training. `./conformer_ctc/train.py --start-epoch 10` loads the checkpoint `./conformer_ctc/exp/epoch-9.pt` and starts training from epoch 10, based on the state from epoch 9.

- `--world-size`

  It is used for multi-GPU single-machine DDP training.

  - (a) If it is 1, then no DDP training is used.
  - (b) If it is 2, then GPU 0 and GPU 1 are used for DDP training.

  The following shows some use cases with it.

  Use case 1: You have 4 GPUs, but you only want to use GPU 0 and GPU 2 for training. You can do the following:

```
$ cd egs/librispeech/ASR
$ export CUDA_VISIBLE_DEVICES="0,2"
$ ./conformer_ctc/train.py --world-size 2
```

Use case 2: You have 4 GPUs and you want to use all of them for training. You can do the following:

```
$ cd egs/librispeech/ASR
$ ./conformer_ctc/train.py --world-size 4
```

Use case 3: You have 4 GPUs but you only want to use GPU 3 for training. You can do the following:

```
$ cd egs/librispeech/ASR
$ export CUDA_VISIBLE_DEVICES="3"
$ ./conformer_ctc/train.py --world-size 1
```

Caution: Only multi-GPU single-machine DDP training is implemented at present. Multi-GPU multi-machine DDP training will be added later.

- `--max-duration`

  It specifies the number of seconds over all utterances in a batch, before **padding**. If you encounter CUDA OOM, please reduce it. For instance, if your are using V100 NVIDIA GPU, we recommend you to set it to `200`.

  Hint: Due to padding, the number of seconds of all utterances in a batch will usually be larger than `--max-duration`.

  A larger value for `--max-duration` may cause OOM during training, while a smaller value may increase the training time. You have to tune it.

Pre-configured options

There are some training options, e.g., weight decay, number of warmup steps, results dir, etc, that are not passed from the commandline. They are pre-configured by the function `get_params()` in conformer_ctc/train.py

You don't need to change these pre-configured parameters. If you really need to change them, please modify `./conformer_ctc/train.py` directly.

Training logs

Training logs and checkpoints are saved in `conformer_ctc/exp`. You will find the following files in that directory:

- `epoch-0.pt`, `epoch-1.pt`, ...

  These are checkpoint files, containing model `state_dict` and optimizer `state_dict`. To resume training from some checkpoint, say `epoch-10.pt`, you can use:

  ```
  $ ./conformer_ctc/train.py --start-epoch 11
  ```

- `tensorboard/`

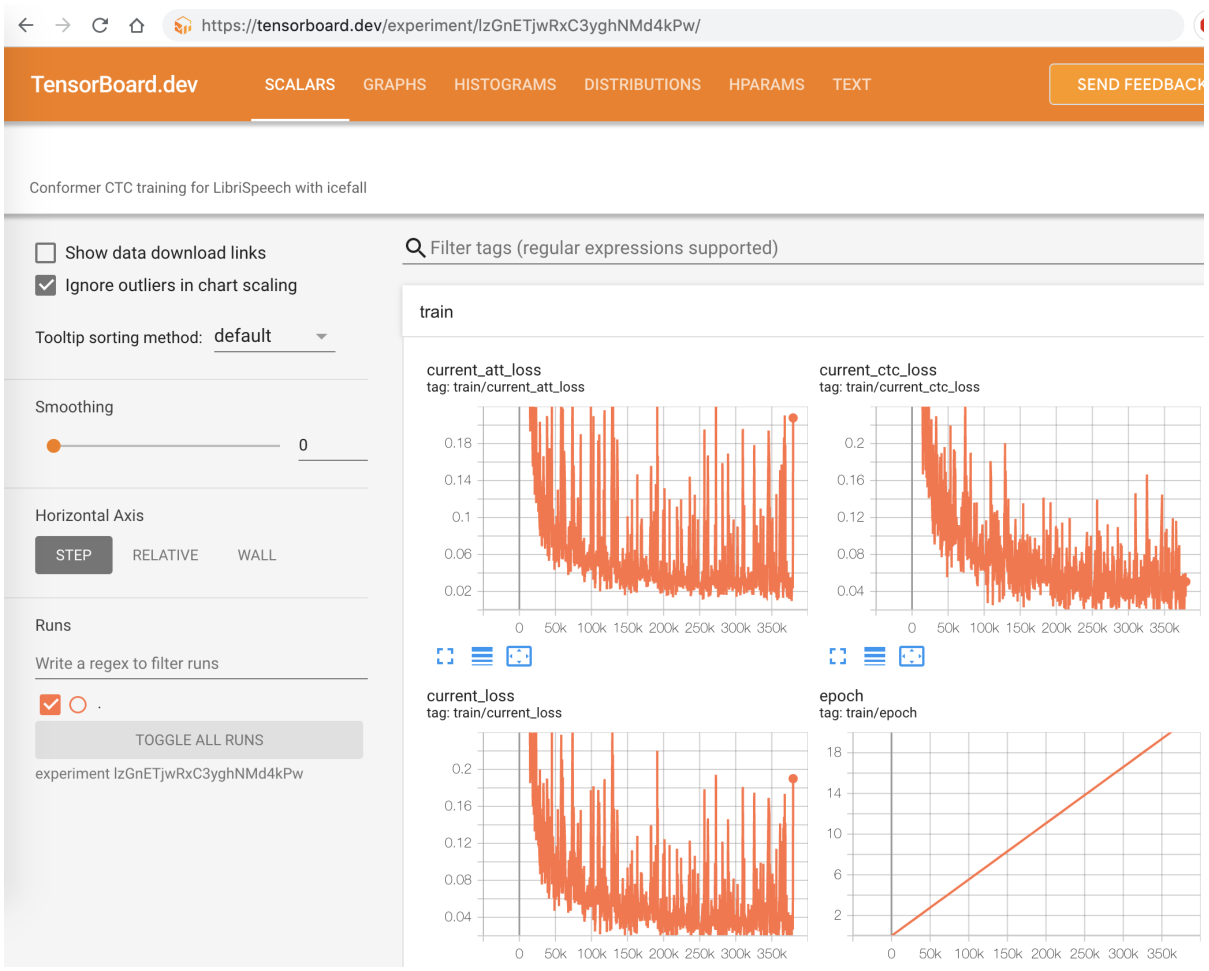
  This folder contains TensorBoard logs. Training loss, validation loss, learning rate, etc, are recorded in these logs. You can visualize them by:

```
$ cd conformer_ctc/exp/tensorboard
$ tensorboard dev upload --logdir . --description "Conformer CTC training␣
↪for LibriSpeech with icefall"
```

It will print something like below:

```
TensorFlow installation not found - running with reduced feature set.
Upload started and will continue reading any new data as it's added to the␣
↪logdir.

To stop uploading, press Ctrl-C.

New experiment created. View your TensorBoard at: https://tensorboard.dev/
↪experiment/lzGnETjwRxC3yghNMd4kPw/

[2021-08-24T16:42:43] Started scanning logdir.
Uploading 4540 scalars...
```

Note there is a URL in the above output, click it and you will see the following screenshot:



Conformer CTC training for LibriSpeech with icefall

☐ Show data download links
☑ Ignore outliers in chart scaling
Tooltip sorting method: default
Smoothing 0

experiment lzGnETjwRxC3yghNMd4kPw

current_att_loss
tag: train/current_att_loss

current_ctc_loss
tag: train/current_ctc_loss

current_loss
tag: train/current_loss

epoch
tag: train/epoch

Fig. 6.4: TensorBoard screenshot.

- `log/log-train-xxxx`

  It is the detailed training log in text format, same as the one you saw printed to the console during training.

Usage examples

The following shows typical use cases:

Case 1

```
$ cd egs/librispeech/ASR
$ ./conformer_ctc/train.py --max-duration 200 --full-libri 0
```

It uses `--max-duration` of 200 to avoid OOM. Also, it uses only a subset of the LibriSpeech data for training.

Case 2

```
$ cd egs/librispeech/ASR
$ export CUDA_VISIBLE_DEVICES="0,3"
$ ./conformer_ctc/train.py --world-size 2
```

It uses GPU 0 and GPU 3 for DDP training.

Case 3

```
$ cd egs/librispeech/ASR
$ ./conformer_ctc/train.py --num-epochs 10 --start-epoch 3
```

It loads checkpoint `./conformer_ctc/exp/epoch-2.pt` and starts training from epoch 3. Also, it trains for 10 epochs.

Decoding

The decoding part uses checkpoints saved by the training part, so you have to run the training part first.

```
$ cd egs/librispeech/ASR
$ ./conformer_ctc/decode.py --help
```

shows the options for decoding.

The commonly used options are:

- `--method`

  This specifies the decoding method. This script supports 7 decoding methods. As for ctc decoding, it uses a sentence piece model to convert word pieces to words. And it needs neither a lexicon nor an n-gram LM.

  For example, the following command uses CTC topology for decoding:

```
$ cd egs/librispeech/ASR
$ ./conformer_ctc/decode.py --method ctc-decoding --max-duration 300
# Caution: The above command is tested with a model with vocab size 500.
```

And the following command uses attention decoder for rescoring:

```
$ cd egs/librispeech/ASR
$ ./conformer_ctc/decode.py --method attention-decoder --max-duration 30 --nbest-
↪scale 0.5
```

- `--nbest-scale`

  It is used to scale down lattice scores so that there are more unique paths for rescoring.

- `--max-duration`

  It has the same meaning as the one during training. A larger value may cause OOM.

Here are some results for CTC decoding with a vocab size of 500:

Usage:

```
$ cd egs/librispeech/ASR
# NOTE: Tested with a model with vocab size 500.
# It won't work for a model with vocab size 5000.
$ ./conformer_ctc/decode.py \
    --epoch 25 \
    --avg 1 \
    --max-duration 300 \
    --exp-dir conformer_ctc/exp \
    --lang-dir data/lang_bpe_500 \
    --method ctc-decoding
```

The output is given below:

```
2021-09-26 12:44:31,033 INFO [decode.py:537] Decoding started
2021-09-26 12:44:31,033 INFO [decode.py:538]
{'lm_dir': PosixPath('data/lm'), 'subsampling_factor': 4, 'vgg_frontend': False, 'use_
↪feat_batchnorm': True,
'feature_dim': 80, 'nhead': 8, 'attention_dim': 512, 'num_decoder_layers': 6, 'search_
↪beam': 20, 'output_beam': 8,
'min_active_states': 30, 'max_active_states': 10000, 'use_double_scores': True,
'epoch': 25, 'avg': 1, 'method': 'ctc-decoding', 'num_paths': 100, 'nbest_scale': 0.5,
'export': False, 'exp_dir': PosixPath('conformer_ctc/exp'), 'lang_dir': PosixPath('data/
↪lang_bpe_500'), 'full_libri': False,
'feature_dir': PosixPath('data/fbank'), 'max_duration': 100, 'bucketing_sampler': False,
↪ 'num_buckets': 30,
'concatenate_cuts': False, 'duration_factor': 1.0, 'gap': 1.0, 'on_the_fly_feats': False,
'shuffle': True, 'return_cuts': True, 'num_workers': 2}
2021-09-26 12:44:31,406 INFO [lexicon.py:113] Loading pre-compiled data/lang_bpe_500/
↪Linv.pt
2021-09-26 12:44:31,464 INFO [decode.py:548] device: cuda:0
2021-09-26 12:44:36,171 INFO [checkpoint.py:92] Loading checkpoint from conformer_ctc/
↪exp/epoch-25.pt
2021-09-26 12:44:36,776 INFO [decode.py:652] Number of model parameters: 109226120
2021-09-26 12:44:37,714 INFO [decode.py:473] batch 0/206, cuts processed until now is 12
```

(continues on next page)

(continued from previous page)

```
2021-09-26 12:45:15,944 INFO [decode.py:473] batch 100/206, cuts processed until now is␣
↪1328
2021-09-26 12:45:54,443 INFO [decode.py:473] batch 200/206, cuts processed until now is␣
↪2563
2021-09-26 12:45:56,411 INFO [decode.py:494] The transcripts are stored in conformer_ctc/
↪exp/recogs-test-clean-ctc-decoding.txt
2021-09-26 12:45:56,592 INFO [utils.py:331] [test-clean-ctc-decoding] %WER 3.26% [1715 /␣
↪52576, 163 ins, 128 del, 1424 sub ]
2021-09-26 12:45:56,807 INFO [decode.py:506] Wrote detailed error stats to conformer_ctc/
↪exp/errs-test-clean-ctc-decoding.txt
2021-09-26 12:45:56,808 INFO [decode.py:522]
For test-clean, WER of different settings are:
ctc-decoding    3.26    best for test-clean

2021-09-26 12:45:57,362 INFO [decode.py:473] batch 0/203, cuts processed until now is 15
2021-09-26 12:46:35,565 INFO [decode.py:473] batch 100/203, cuts processed until now is␣
↪1477
2021-09-26 12:47:15,106 INFO [decode.py:473] batch 200/203, cuts processed until now is␣
↪2922
2021-09-26 12:47:16,131 INFO [decode.py:494] The transcripts are stored in conformer_ctc/
↪exp/recogs-test-other-ctc-decoding.txt
2021-09-26 12:47:16,208 INFO [utils.py:331] [test-other-ctc-decoding] %WER 8.21% [4295 /␣
↪52343, 396 ins, 315 del, 3584 sub ]
2021-09-26 12:47:16,432 INFO [decode.py:506] Wrote detailed error stats to conformer_ctc/
↪exp/errs-test-other-ctc-decoding.txt
2021-09-26 12:47:16,432 INFO [decode.py:522]
For test-other, WER of different settings are:
ctc-decoding    8.21    best for test-other

2021-09-26 12:47:16,433 INFO [decode.py:680] Done!
```

Pre-trained Model

We have uploaded a pre-trained model to https://huggingface.co/csukuangfj/icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09

We describe how to use the pre-trained model to transcribe a sound file or multiple sound files in the following.

Install kaldifeat

kaldifeat is used to extract features for a single sound file or multiple sound files at the same time.

Please refer to https://github.com/csukuangfj/kaldifeat for installation.

Download the pre-trained model

The following commands describe how to download the pre-trained model:

```
$ cd egs/librispeech/ASR
$ git clone https://huggingface.co/csukuangfj/icefall-asr-librispeech-conformer-ctc-jit-
↪bpe-500-2021-11-09
$ cd icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09
$ git lfs pull
```

Caution: You have to use `git lfs pull` to download the pre-trained model. Otherwise, you will have the following issue when running `decode.py`:

```
_pickle.UnpicklingError: invalid load key, 'v'
```

To fix that issue, please use:

```
cd icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09
git lfs pull
```

Caution: In order to use this pre-trained model, your k2 version has to be v1.9 or later.

After downloading, you will have the following files:

```
$ cd egs/librispeech/ASR
$ tree icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09
```

```
icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09
|-- README.md
|-- data
|   |-- lang_bpe_500
|   |   |-- HLG.pt
|   |   |-- HLG_modified.pt
|   |   |-- bpe.model
|   |   |-- tokens.txt
|   |   `-- words.txt
|   `-- lm
|       `-- G_4_gram.pt
|-- exp
|   |-- cpu_jit.pt
|   `-- pretrained.pt
|-- log
|   `-- log-decode-2021-11-09-17-38-28
`-- test_wavs
```

(continues on next page)

(continued from previous page)

```
    |-- 1089-134686-0001.wav
    |-- 1221-135766-0001.wav
    |-- 1221-135766-0002.wav
    `-- trans.txt
```

File descriptions:

- `data/lang_bpe_500/HLG.pt`

  It is the decoding graph.

- `data/lang_bpe_500/HLG_modified.pt`

  It uses a modified CTC topology while building HLG.

- `data/lang_bpe_500/bpe.model`

  It is a sentencepiece model. You can use it to reproduce our results.

- `data/lang_bpe_500/tokens.txt`

  It contains tokens and their IDs, generated from `bpe.model`. Provided only for convenience so that you can look up the SOS/EOS ID easily.

- `data/lang_bpe_500/words.txt`

  It contains words and their IDs.

- `data/lm/G_4_gram.pt`

  It is a 4-gram LM, used for n-gram LM rescoring.

- `exp/pretrained.pt`

  It contains pre-trained model parameters, obtained by averaging checkpoints from `epoch-23.pt` to `epoch-77.pt`. Note: We have removed optimizer `state_dict` to reduce file size.

- `exp/cpu_jit.pt`

  It contains torch scripted model that can be deployed in C++.

- `test_wavs/*.wav`

  It contains some test sound files from LibriSpeech `test-clean` dataset.

- `test_wavs/trans.txt`

  It contains the reference transcripts for the sound files in `test_wavs/`.

The information of the test sound files is listed below:

```
$ soxi icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/test_wavs/*.wav

Input File     : 'icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/test_wavs/
↪1089-134686-0001.wav'
Channels       : 1
Sample Rate    : 16000
Precision      : 16-bit
Duration       : 00:00:06.62 = 106000 samples ~ 496.875 CDDA sectors
File Size      : 212k
Bit Rate       : 256k
Sample Encoding: 16-bit Signed Integer PCM
```

(continues on next page)

(continued from previous page)

```
Input File     : 'icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/test_wavs/
↪1221-135766-0001.wav'
Channels       : 1
Sample Rate    : 16000
Precision      : 16-bit
Duration       : 00:00:16.71 = 267440 samples ~ 1253.62 CDDA sectors
File Size      : 535k
Bit Rate       : 256k
Sample Encoding: 16-bit Signed Integer PCM


Input File     : 'icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/test_wavs/
↪1221-135766-0002.wav'
Channels       : 1
Sample Rate    : 16000
Precision      : 16-bit
Duration       : 00:00:04.83 = 77200 samples ~ 361.875 CDDA sectors
File Size      : 154k
Bit Rate       : 256k
Sample Encoding: 16-bit Signed Integer PCM

Total Duration of 3 files: 00:00:28.16
```

Usage

```
$ cd egs/librispeech/ASR
$ ./conformer_ctc/pretrained.py --help
```

displays the help information.

It supports 4 decoding methods:

- CTC decoding
- HLG decoding
- HLG + n-gram LM rescoring
- HLG + n-gram LM rescoring + attention decoder rescoring

CTC decoding

CTC decoding uses the best path of the decoding lattice as the decoding result without any LM or lexicon.

The command to run CTC decoding is:

```
$ cd egs/librispeech/ASR
$ ./conformer_ctc/pretrained.py \
   --checkpoint ./icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/exp/
↪pretrained.pt \
   --bpe-model ./icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/data/lang_
```

(continues on next page)

(continued from previous page)

```
↪bpe_500/bpe.model \
  --method ctc-decoding \
  --num-classes 500 \
  ./icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/test_wavs/1089-134686-
↪0001.wav \
  ./icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/test_wavs/1221-135766-
↪0001.wav \
  ./icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/test_wavs/1221-135766-
↪0002.wav
```

The output is given below:

```
2021-11-10 12:12:29,554 INFO [pretrained.py:260] {'sample_rate': 16000, 'subsampling_
↪factor': 4, 'vgg_frontend': False, 'use_feat_batchnorm': True, 'feature_dim': 80,
↪'nhead': 8, 'attention_dim': 512, 'num_decoder_layers': 0, 'search_beam': 20, 'output_
↪beam': 8, 'min_active_states': 30, 'max_active_states': 10000, 'use_double_scores':␣
↪True, 'checkpoint': './icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/
↪exp/pretrained.pt', 'words_file': None, 'HLG': None, 'bpe_model': './icefall-asr-
↪librispeech-conformer-ctc-jit-bpe-500-2021-11-09/data/lang_bpe_500/bpe.model', 'method
↪': 'ctc-decoding', 'G': None, 'num_paths': 100, 'ngram_lm_scale': 1.3, 'attention_
↪decoder_scale': 1.2, 'nbest_scale': 0.5, 'sos_id': 1, 'num_classes': 500, 'eos_id': 1,
↪'sound_files': ['./icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/test_
↪wavs/1089-134686-0001.wav', './icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-
↪11-09/test_wavs/1221-135766-0001.wav', './icefall-asr-librispeech-conformer-ctc-jit-
↪bpe-500-2021-11-09/test_wavs/1221-135766-0002.wav'], 'env_info': {'k2-version': '1.9',
↪'k2-build-type': 'Release', 'k2-with-cuda': True, 'k2-git-sha1':
↪'7178d67e594bc7fa89c2b331ad7bd1c62a6a9eb4', 'k2-git-date': 'Tue Oct 26 22:12:54 2021',
↪'lhotse-version': '0.11.0.dev+missing.version.file', 'torch-cuda-available': True,
↪'torch-cuda-version': '10.1', 'python-version': '3.8', 'icefall-git-branch': 'bpe-500',
↪ 'icefall-git-sha1': '8d93169-dirty', 'icefall-git-date': 'Wed Nov 10 11:52:44 2021',
↪'icefall-path': '/ceph-fj/fangjun/open-source-2/icefall-fix', 'k2-path': '/ceph-fj/
↪fangjun/open-source-2/k2-bpe-500/k2/python/k2/__init__.py', 'lhotse-path': '/ceph-fj/
↪fangjun/open-source-2/lhotse-bpe-500/lhotse/__init__.py'}}
2021-11-10 12:12:29,554 INFO [pretrained.py:266] device: cuda:0
2021-11-10 12:12:29,554 INFO [pretrained.py:268] Creating model
2021-11-10 12:12:35,600 INFO [pretrained.py:285] Constructing Fbank computer
2021-11-10 12:12:35,601 INFO [pretrained.py:295] Reading sound files: ['./icefall-asr-
↪librispeech-conformer-ctc-jit-bpe-500-2021-11-09/test_wavs/1089-134686-0001.wav', './
↪icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/test_wavs/1221-135766-
↪0001.wav', './icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/test_wavs/
↪1221-135766-0002.wav']
2021-11-10 12:12:35,758 INFO [pretrained.py:301] Decoding started
2021-11-10 12:12:36,025 INFO [pretrained.py:319] Use CTC decoding
2021-11-10 12:12:36,204 INFO [pretrained.py:425]
./icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/test_wavs/1089-134686-
↪0001.wav:
AFTER EARLY NIGHTFALL THE YELLOW LAMPS WOULD LIGHT UP HERE AND THERE THE SQUALID QUARTER␣
↪OF THE BROFFELS

./icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/test_wavs/1221-135766-
↪0001.wav:
GOD AS A DIRECT CONSEQUENCE OF THE SIN WHICH MAN THUS PUNISHED HAD GIVEN HER A LOVELY␣
```

(continues on next page)

(continued from previous page)

```
↪CHILD WHOSE PLACE WAS ON THAT SAME DISHONORED B
OSOM TO CONNECT HER PARENT FOREVER WITH THE RACE AND DESCENT OF MORTALS AND TO BE␣
↪FINALLY A BLESSED SOUL IN HEAVEN

./icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/test_wavs/1221-135766-
↪0002.wav:
YET THESE THOUGHTS AFFECTED HESTER PRYNNE LESS WITH HOPE THAN APPREHENSION

2021-11-10 12:12:36,204 INFO [pretrained.py:427] Decoding Done
```

HLG decoding

HLG decoding uses the best path of the decoding lattice as the decoding result.

The command to run HLG decoding is:

```
$ cd egs/librispeech/ASR
$ ./conformer_ctc/pretrained.py \
   --checkpoint ./icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/exp/
↪pretrained.pt \
   --words-file ./icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/data/lang_
↪bpe_500/words.txt \
   --method 1best \
   --num-classes 500 \
   --HLG ./icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/data/lang_bpe_
↪500/HLG.pt \
   ./icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/test_wavs/1089-134686-
↪0001.wav \
   ./icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/test_wavs/1221-135766-
↪0001.wav \
   ./icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/test_wavs/1221-135766-
↪0002.wav
```

The output is given below:

```
2021-11-10 13:33:03,723 INFO [pretrained.py:260] {'sample_rate': 16000, 'subsampling_
↪factor': 4, 'vgg_frontend': False, 'use_feat_batchnorm': True, 'feature_dim': 80,
↪'nhead': 8, 'attention_dim': 512, 'num_decoder_layers': 0, 'search_beam': 20, 'output_
↪beam': 8, 'min_active_states': 30, 'max_active_states': 10000, 'use_double_scores':␣
↪True, 'checkpoint': './icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/
↪exp/pretrained.pt', 'words_file': './icefall-asr-librispeech-conformer-ctc-jit-bpe-500-
↪2021-11-09/data/lang_bpe_500/words.txt', 'HLG': './icefall-asr-librispeech-conformer-
↪ctc-jit-bpe-500-2021-11-09/data/lang_bpe_500/HLG.pt', 'bpe_model': None, 'method':
↪'1best', 'G': None, 'num_paths': 100, 'ngram_lm_scale': 1.3, 'attention_decoder_scale
↪': 1.2, 'nbest_scale': 0.5, 'sos_id': 1, 'num_classes': 500, 'eos_id': 1, 'sound_files
↪': ['./icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/test_wavs/1089-
↪134686-0001.wav', './icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/test_
↪wavs/1221-135766-0001.wav', './icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-
↪11-09/test_wavs/1221-135766-0002.wav'], 'env_info': {'k2-version': '1.9', 'k2-build-
↪type': 'Release', 'k2-with-cuda': True, 'k2-git-sha1':
↪'7178d67e594bc7fa89c2b331ad7bd1c62a6a9eb4', 'k2-git-date': 'Tue Oct 26 22:12:54 2021',
```

(continues on next page)

(continued from previous page)

```
↪'lhotse-version': '0.11.0.dev+missing.version.file', 'torch-cuda-available': True,
↪'torch-cuda-version': '10.1', 'python-version': '3.8', 'icefall-git-branch': 'bpe-500',
↪ 'icefall-git-sha1': '8d93169-dirty', 'icefall-git-date': 'Wed Nov 10 11:52:44 2021',
↪'icefall-path': '/ceph-fj/fangjun/open-source-2/icefall-fix', 'k2-path': '/ceph-fj/
↪fangjun/open-source-2/k2-bpe-500/k2/python/k2/__init__.py', 'lhotse-path': '/ceph-fj/
↪fangjun/open-source-2/lhotse-bpe-500/lhotse/__init__.py'}}
2021-11-10 13:33:03,723 INFO [pretrained.py:266] device: cuda:0
2021-11-10 13:33:03,723 INFO [pretrained.py:268] Creating model
2021-11-10 13:33:09,775 INFO [pretrained.py:285] Constructing Fbank computer
2021-11-10 13:33:09,776 INFO [pretrained.py:295] Reading sound files: ['./icefall-asr-
↪librispeech-conformer-ctc-jit-bpe-500-2021-11-09/test_wavs/1089-134686-0001.wav', './
↪icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/test_wavs/1221-135766-
↪0001.wav', './icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/test_wavs/
↪1221-135766-0002.wav']
2021-11-10 13:33:09,881 INFO [pretrained.py:301] Decoding started
2021-11-10 13:33:09,951 INFO [pretrained.py:352] Loading HLG from ./icefall-asr-
↪librispeech-conformer-ctc-jit-bpe-500-2021-11-09/data/lang_bpe_500/HLG.pt
2021-11-10 13:33:13,234 INFO [pretrained.py:384] Use HLG decoding
2021-11-10 13:33:13,571 INFO [pretrained.py:425]
./icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/test_wavs/1089-134686-
↪0001.wav:
AFTER EARLY NIGHTFALL THE YELLOW LAMPS WOULD LIGHT UP HERE AND THERE THE SQUALID QUARTER␣
↪OF THE BROTHELS

./icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/test_wavs/1221-135766-
↪0001.wav:
GOD AS A DIRECT CONSEQUENCE OF THE SIN WHICH MAN THUS PUNISHED HAD GIVEN HER A LOVELY␣
↪CHILD WHOSE PLACE WAS ON THAT SAME DISHONORED BOSOM TO CONNECT HER PARENT FOREVER WITH␣
↪THE RACE AND DESCENT OF MORTALS AND TO BE FINALLY A BLESSED SOUL IN HEAVEN

./icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/test_wavs/1221-135766-
↪0002.wav:
YET THESE THOUGHTS AFFECTED HESTER PRYNNE LESS WITH HOPE THAN APPREHENSION

2021-11-10 13:33:13,571 INFO [pretrained.py:427] Decoding Done
```

HLG decoding + LM rescoring

It uses an n-gram LM to rescore the decoding lattice and the best path of the rescored lattice is the decoding result.

The command to run HLG decoding + LM rescoring is:

```
$ cd egs/librispeech/ASR
./conformer_ctc/pretrained.py \
   --checkpoint ./icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/exp/
↪pretrained.pt \
   --words-file ./icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/data/lang_
↪bpe_500/words.txt \
   --method whole-lattice-rescoring \
   --num-classes 500 \
   --HLG ./icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/data/lang_bpe_
```

(continues on next page)

(continued from previous page)

```
↪500/HLG.pt \
  --G ./icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/data/lm/G_4_gram.
↪pt \
  --ngram-lm-scale 1.0 \
  ./icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/test_wavs/1089-134686-
↪0001.wav \
  ./icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/test_wavs/1221-135766-
↪0001.wav \
  ./icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/test_wavs/1221-135766-
↪0002.wav
```

Its output is:

```
2021-11-10 13:39:55,857 INFO [pretrained.py:260] {'sample_rate': 16000, 'subsampling_
↪factor': 4, 'vgg_frontend': False, 'use_feat_batchnorm': True, 'feature_dim': 80,
↪'nhead': 8, 'attention_dim': 512, 'num_decoder_layers': 0, 'search_beam': 20, 'output_
↪beam': 8, 'min_active_states': 30, 'max_active_states': 10000, 'use_double_scores':␣
↪True, 'checkpoint': './icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/
↪exp/pretrained.pt', 'words_file': './icefall-asr-librispeech-conformer-ctc-jit-bpe-500-
↪2021-11-09/data/lang_bpe_500/words.txt', 'HLG': './icefall-asr-librispeech-conformer-
↪ctc-jit-bpe-500-2021-11-09/data/lang_bpe_500/HLG.pt', 'bpe_model': None, 'method':
↪'whole-lattice-rescoring', 'G': './icefall-asr-librispeech-conformer-ctc-jit-bpe-500-
↪2021-11-09/data/lm/G_4_gram.pt', 'num_paths': 100, 'ngram_lm_scale': 1.0, 'attention_
↪decoder_scale': 1.2, 'nbest_scale': 0.5, 'sos_id': 1, 'num_classes': 500, 'eos_id': 1,
↪'sound_files': ['./icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/test_
↪wavs/1089-134686-0001.wav', './icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-
↪11-09/test_wavs/1221-135766-0001.wav', './icefall-asr-librispeech-conformer-ctc-jit-
↪bpe-500-2021-11-09/test_wavs/1221-135766-0002.wav'], 'env_info': {'k2-version': '1.9',
↪'k2-build-type': 'Release', 'k2-with-cuda': True, 'k2-$it-sha1':
↪'7178d67e594bc7fa89c2b331ad7bd1c62a6a9eb4', 'k2-git-date': 'Tue Oct 26 22:12:54 2021',
↪'lhotse-version': '0.11.0.dev+missing.version.file', 'torch-cuda-available': True,
↪'torch-cuda-version': '10.1', 'python-version': '3.8', 'icefall-git-branch': 'bpe-500',
↪ 'icefall-git-sha1': '8d93169-dirty', 'icefall-git-date': 'Wed Nov 10 11:52:44 2021',
↪'icefall-path': '/ceph-fj/fangjun/open-source-2/icefall-fix', 'k2-path': '/ceph-fj/
↪fangjun/open-source-2/k2-bpe-500/k2/python/k2/__init__.py', 'lhotse-path': '/ceph-fj/
↪fangjun/open-source-2/lhotse-bpe-500/lhotse/__init__.py'}}
2021-11-10 13:39:55,858 INFO [pretrained.py:266] device: cuda:0
2021-11-10 13:39:55,858 INFO [pretrained.py:268] Creating model
2021-11-10 13:40:01,979 INFO [pretrained.py:285] Constructing Fbank computer
2021-11-10 13:40:01,980 INFO [pretrained.py:295] Reading sound files: ['./icefall-asr-
↪librispeech-conformer-ctc-jit-bpe-500-2021-11-09/test_wavs/1089-134686-0001.wav', './
↪icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/test_wavs/1221-135766-
↪0001.wav', './icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/test_wavs/
↪1221-135766-0002.wav']
2021-11-10 13:40:02,055 INFO [pretrained.py:301] Decoding started
2021-11-10 13:40:02,117 INFO [pretrained.py:352] Loading HLG from ./icefall-asr-
↪librispeech-conformer-ctc-jit-bpe-500-2021-11-09/data/lang_bpe_500/HLG.pt
2021-11-10 13:40:05,051 INFO [pretrained.py:363] Loading G from ./icefall-asr-
↪librispeech-conformer-ctc-jit-bpe-500-2021-11-09/data/lm/G_4_gram.pt
2021-11-10 13:40:18,959 INFO [pretrained.py:389] Use HLG decoding + LM rescoring
2021-11-10 13:40:19,546 INFO [pretrained.py:425]
./icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/test_wavs/1089-134686-
```

(continues on next page)

(continued from previous page)

```
↪0001.wav:
AFTER EARLY NIGHTFALL THE YELLOW LAMPS WOULD LIGHT UP HERE AND THERE THE SQUALID QUARTER␣
↪OF THE BROTHELS

./icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/test_wavs/1221-135766-
↪0001.wav:
GOD AS A DIRECT CONSEQUENCE OF THE SIN WHICH MAN THUS PUNISHED HAD GIVEN HER A LOVELY␣
↪CHILD WHOSE PLACE WAS ON THAT SAME DISHONORED BOSOM TO CONNECT HER PARENT FOREVER WITH␣
↪THE RACE AND DESCENT OF MORTALS AND TO BE FINALLY A BLESSED SOUL IN HEAVEN

./icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/test_wavs/1221-135766-
↪0002.wav:
YET THESE THOUGHTS AFFECTED HESTER PRYNNE LESS WITH HOPE THAN APPREHENSION

2021-11-10 13:40:19,546 INFO [pretrained.py:427] Decoding Done
```

HLG decoding + LM rescoring + attention decoder rescoring

It uses an n-gram LM to rescore the decoding lattice, extracts n paths from the rescored lattice, recores the extracted paths with an attention decoder. The path with the highest score is the decoding result.

The command to run HLG decoding + LM rescoring + attention decoder rescoring is:

```
$ cd egs/librispeech/ASR
$ ./conformer_ctc/pretrained.py \
  --checkpoint ./icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/exp/
↪pretrained.pt \
  --words-file ./icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/data/lang_
↪bpe_500/words.txt \
  --method attention-decoder \
  --num-classes 500 \
  --HLG ./icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/data/lang_bpe_
↪500/HLG.pt \
  --G ./icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/data/lm/G_4_gram.
↪pt \
  --ngram-lm-scale 2.0 \
  --attention-decoder-scale 2.0 \
  --nbest-scale 0.5 \
  --num-paths 100 \
  --sos-id 1 \
  --eos-id 1 \
  ./icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/test_wavs/1089-134686-
↪0001.wav \
  ./icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/test_wavs/1221-135766-
↪0001.wav \
  ./icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/test_wavs/1221-135766-
↪0002.wav
```

The output is below:

```
2021-11-10 13:43:45,598 INFO [pretrained.py:260] {'sample_rate': 16000, 'subsampling_
↪factor': 4, 'vgg_frontend': False, 'use_feat_batchnorm': True, 'feature_dim': 80,
↪'nhead': 8, 'attention_dim': 512, 'num_decoder_layers': 6, 'search_beam': 20, 'output_
↪beam': 8, 'min_active_states': 30, 'max_active_states': 10000, 'use_double_scores':␣
↪True, 'checkpoint': './icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/
↪exp/pretrained.pt', 'words_file': './icefall-asr-librispeech-conformer-ctc-jit-bpe-500-
↪2021-11-09/data/lang_bpe_500/words.txt', 'HLG': './icefall-asr-librispeech-conformer-
↪ctc-jit-bpe-500-2021-11-09/data/lang_bpe_500/HLG.pt', 'bpe_model': None, 'method':
↪'attention-decoder', 'G': './icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-
↪09/data/lm/G_4_gram.pt', 'num_paths': 100, 'ngram_lm_scale': 2.0, 'attention_decoder_
↪scale': 2.0, 'nbest_scale': 0.5, 'sos_id': 1, 'num_classes': 500, 'eos_id': 1, 'sound_
↪files': ['./icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/test_wavs/
↪1089-134686-0001.wav', './icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/
↪test_wavs/1221-135766-0001.wav', './icefall-asr-librispeech-conformer-ctc-jit-bpe-500-
↪2021-11-09/test_wavs/1221-135766-0002.wav'], 'env_info': {'k2-version': '1.9', 'k2-
↪build-type': 'Release', 'k2-with-cuda': True, 'k2-git-sha1':
↪'7178d67e594bc7fa89c2b331ad7bd1c62a6a9eb4', 'k2-git-date': 'Tue Oct 26 22:12:54 2021',
↪'lhotse-version': '0.11.0.dev+missing.version.file', 'torch-cuda-available': True,
↪'torch-cuda-version': '10.1', 'python-version': '3.8', 'icefall-git-branch': 'bpe-500',
↪ 'icefall-git-sha1': '8d93169-dirty', 'icefall-git-date': 'Wed Nov 10 11:52:44 2021',
↪'icefall-path': '/ceph-fj/fangjun/open-source-2/icefall-fix', 'k2-path': '/ceph-fj/
↪fangjun/open-source-2/k2-bpe-500/k2/python/k2/__init__.py', 'lhotse-path': '/ceph-fj/
↪fangjun/open-source-2/lhotse-bpe-500/lhotse/__init__.py'}}
2021-11-10 13:43:45,599 INFO [pretrained.py:266] device: cuda:0
2021-11-10 13:43:45,599 INFO [pretrained.py:268] Creating model
2021-11-10 13:43:51,833 INFO [pretrained.py:285] Constructing Fbank computer
2021-11-10 13:43:51,834 INFO [pretrained.py:295] Reading sound files: ['./icefall-asr-
↪librispeech-conformer-ctc-jit-bpe-500-2021-11-09/test_wavs/1089-134686-0001.wav', './
↪icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/test_wavs/1221-135766-
↪0001.wav', './icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/test_wavs/
↪1221-135766-0002.wav']
2021-11-10 13:43:51,915 INFO [pretrained.py:301] Decoding started
2021-11-10 13:43:52,076 INFO [pretrained.py:352] Loading HLG from ./icefall-asr-
↪librispeech-conformer-ctc-jit-bpe-500-2021-11-09/data/lang_bpe_500/HLG.pt
2021-11-10 13:43:55,110 INFO [pretrained.py:363] Loading G from ./icefall-asr-
↪librispeech-conformer-ctc-jit-bpe-500-2021-11-09/data/lm/G_4_gram.pt
2021-11-10 13:44:09,329 INFO [pretrained.py:397] Use HLG + LM rescoring + attention␣
↪decoder rescoring
2021-11-10 13:44:10,192 INFO [pretrained.py:425]
./icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/test_wavs/1089-134686-
↪0001.wav:
AFTER EARLY NIGHTFALL THE YELLOW LAMPS WOULD LIGHT UP HERE AND THERE THE SQUALID QUARTER␣
↪OF THE BROTHELS

./icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/test_wavs/1221-135766-
↪0001.wav:
GOD AS A DIRECT CONSEQUENCE OF THE SIN WHICH MAN THUS PUNISHED HAD GIVEN HER A LOVELY␣
↪CHILD WHOSE PLACE WAS ON THAT SAME DISHONORED BOSOM TO CONNECT HER PARENT FOREVER WITH␣
↪THE RACE AND DESCENT OF MORTALS AND TO BE FINALLY A BLESSED SOUL IN HEAVEN

./icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/test_wavs/1221-135766-
↪0002.wav:
YET THESE THOUGHTS AFFECTED HESTER PRYNNE LESS WITH HOPE THAN APPREHENSION
```

(continues on next page)

(continued from previous page)

```
2021-11-10 13:44:10,192 INFO [pretrained.py:427] Decoding Done
```

Compute WER with the pre-trained model

To check the WER of the pre-trained model on the test datasets, run:

```
$ cd egs/librispeech/ASR
$ cd icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/exp/
$ ln -s pretrained.pt epoch-999.pt
$ cd ../..
$ ./conformer_ctc/decode.py \
    --exp-dir ./icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/exp \
    --lang-dir ./icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/data/lang_
↪bpe_500 \
    --lm-dir ./icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/data/lm \
    --epoch 999 \
    --avg 1 \
    --concatenate-cuts 0 \
    --bucketing-sampler 1 \
    --max-duration 30 \
    --num-paths 1000 \
    --method attention-decoder \
    --nbest-scale 0.5
```

Colab notebook

We do provide a colab notebook for this recipe showing how to use a pre-trained model.

Hint: Due to limited memory provided by Colab, you have to upgrade to Colab Pro to run `HLG decoding + LM rescoring` and `HLG decoding + LM rescoring + attention decoder rescoring`. Otherwise, you can only run `HLG decoding` with Colab.

Congratulations! You have finished the LibriSpeech ASR recipe with conformer CTC models in `icefall`.

If you want to deploy your trained model in C++, please read the following section.

Deployment with C++

This section describes how to deploy the pre-trained model in C++, without Python dependencies.

Hint: At present, it does NOT support streaming decoding.

First, let us compile k2 from source:

```
$ cd $HOME
$ git clone https://github.com/k2-fsa/k2
$ cd k2
$ git checkout v2.0-pre
```

Caution: You have to switch to the branch `v2.0-pre`!

```
$ mkdir build-release
$ cd build-release
$ cmake -DCMAKE_BUILD_TYPE=Release ..
$ make -j ctc_decode hlg_decode ngram_lm_rescore attention_rescore

# You will find four binaries in `./bin`, i.e.,
# ./bin/ctc_decode, ./bin/hlg_decode,
# ./bin/ngram_lm_rescore, and ./bin/attention_rescore
```

Now you are ready to go!

Assume you have run:

```
$ cd k2/build-release
$ ln -s /path/to/icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09 .
↪/
```

To view the usage of `./bin/ctc_decode`, run:

```
$ ./bin/ctc_decode
```

It will show you the following message:

```
Please provide --nn_model

This file implements decoding with a CTC topology, without any
kinds of LM or lexicons.

Usage:
  ./bin/ctc_decode \
    --use_gpu true \
    --nn_model <path to torch scripted pt file> \
    --bpe_model <path to pre-trained BPE model> \
    <path to foo.wav> \
    <path to bar.wav> \
    <more waves if any>

To see all possible options, use
  ./bin/ctc_decode --help

Caution:
 - Only sound files (*.wav) with single channel are supported.
 - It assumes the model is conformer_ctc/transformer.py from icefall.
   If you use a different model, you have to change the code
   related to `model.forward` in this file.
```

CTC decoding

```
./bin/ctc_decode \
  --use_gpu true \
  --nn_model ./icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/exp/cpu_jit.
↪pt \
  --bpe_model ./icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/data/lang_
↪bpe_500/bpe.model \
  ./icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/test_wavs/1089-134686-
↪0001.wav \
  ./icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/test_wavs/1221-135766-
↪0001.wav \
  ./icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/test_wavs/1221-135766-
↪0002.wav
```

Its output is:

```
2021-11-10 13:57:55.316 [I] k2/torch/bin/ctc_decode.cu:105:int main(int, char**) Use GPU
2021-11-10 13:57:55.316 [I] k2/torch/bin/ctc_decode.cu:109:int main(int, char**) Device:␣
↪cuda:0
2021-11-10 13:57:55.316 [I] k2/torch/bin/ctc_decode.cu:118:int main(int, char**) Load␣
↪wave files
2021-11-10 13:58:01.221 [I] k2/torch/bin/ctc_decode.cu:125:int main(int, char**) Build␣
↪Fbank computer
2021-11-10 13:58:01.222 [I] k2/torch/bin/ctc_decode.cu:136:int main(int, char**) Compute␣
↪features
2021-11-10 13:58:01.228 [I] k2/torch/bin/ctc_decode.cu:144:int main(int, char**) Load␣
↪neural network model
2021-11-10 13:58:02.19 [I] k2/torch/bin/ctc_decode.cu:159:int main(int, char**) Compute␣
↪nnet_output
2021-11-10 13:58:02.543 [I] k2/torch/bin/ctc_decode.cu:174:int main(int, char**) Build␣
↪CTC topo
2021-11-10 13:58:02.547 [I] k2/torch/bin/ctc_decode.cu:177:int main(int, char**) Decoding
2021-11-10 13:58:02.708 [I] k2/torch/bin/ctc_decode.cu:207:int main(int, char**)
Decoding result:

./icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/test_wavs/1089-134686-
↪0001.wav
AFTER EARLY NIGHTFALL THE YELLOW LAMPS WOULD LIGHT UP HERE AND THERE THE SQUALID QUARTER␣
↪OF THE BROFFELS

./icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/test_wavs/1221-135766-
↪0001.wav
GOD AS A DIRECT CONSEQUENCE OF THE SIN WHICH MAN THUS PUNISHED HAD GIVEN HER A LOVELY␣
↪CHILD WHOSE PLACE WAS ON THAT SAME DISHONORED BOSOM TO CONNECT HER PARENT FOREVER WITH␣
↪THE RACE AND DESCENT OF MORTALS AND TO BE FINALLY A BLESSED SOUL IN HEAVEN

./icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/test_wavs/1221-135766-
↪0002.wav
YET THESE THOUGHTS AFFECTED HESTER PRYNNE LESS WITH HOPE THAN APPREHENSION
```

HLG decoding

```
./bin/hlg_decode \
  --use_gpu true \
  --nn_model ./icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/exp/cpu_jit.
↪pt \
  --hlg ./icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/data/lang_bpe_500/
↪HLG.pt \
  --word_table ./icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/data/lang_
↪bpe_500/words.txt \
  ./icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/test_wavs/1089-134686-
↪0001.wav \
  ./icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/test_wavs/1221-135766-
↪0001.wav \
  ./icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/test_wavs/1221-135766-
↪0002.wav
```

The output is:

```
2021-11-10 13:59:04.729 [I] k2/torch/bin/hlg_decode.cu:111:int main(int, char**) Use GPU
2021-11-10 13:59:04.729 [I] k2/torch/bin/hlg_decode.cu:115:int main(int, char**) Device:␣
↪cuda:0
2021-11-10 13:59:04.729 [I] k2/torch/bin/hlg_decode.cu:124:int main(int, char**) Load␣
↪wave files
2021-11-10 13:59:10.702 [I] k2/torch/bin/hlg_decode.cu:131:int main(int, char**) Build␣
↪Fbank computer
2021-11-10 13:59:10.703 [I] k2/torch/bin/hlg_decode.cu:142:int main(int, char**) Compute␣
↪features
2021-11-10 13:59:10.707 [I] k2/torch/bin/hlg_decode.cu:150:int main(int, char**) Load␣
↪neural network model
2021-11-10 13:59:11.545 [I] k2/torch/bin/hlg_decode.cu:165:int main(int, char**) Compute␣
↪nnet_output
2021-11-10 13:59:12.72 [I] k2/torch/bin/hlg_decode.cu:180:int main(int, char**) Load ./
↪icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/data/lang_bpe_500/HLG.pt
2021-11-10 13:59:12.994 [I] k2/torch/bin/hlg_decode.cu:185:int main(int, char**) Decoding
2021-11-10 13:59:13.268 [I] k2/torch/bin/hlg_decode.cu:216:int main(int, char**)
Decoding result:

./icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/test_wavs/1089-134686-
↪0001.wav
AFTER EARLY NIGHTFALL THE YELLOW LAMPS WOULD LIGHT UP HERE AND THERE THE SQUALID QUARTER␣
↪OF THE BROTHELS

./icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/test_wavs/1221-135766-
↪0001.wav
GOD AS A DIRECT CONSEQUENCE OF THE SIN WHICH MAN THUS PUNISHED HAD GIVEN HER A LOVELY␣
↪CHILD WHOSE PLACE WAS ON THAT SAME DISHONORED BOSOM TO CONNECT HER PARENT FOREVER WITH␣
↪THE RACE AND DESCENT OF MORTALS AND TO BE FINALLY A BLESSED SOUL IN HEAVEN

./icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/test_wavs/1221-135766-
↪0002.wav
YET THESE THOUGHTS AFFECTED HESTER PRYNNE LESS WITH HOPE THAN APPREHENSION
```

HLG decoding + n-gram LM rescoring

```
./bin/ngram_lm_rescore \
  --use_gpu true \
  --nn_model ./icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/exp/cpu_jit.
↪pt \
  --hlg ./icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/data/lang_bpe_500/
↪HLG.pt \
  --g ./icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/data/lm/G_4_gram.pt␣
↪\
  --ngram_lm_scale 1.0 \
  --word_table ./icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/data/lang_
↪bpe_500/words.txt \
  ./icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/test_wavs/1089-134686-
↪0001.wav \
  ./icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/test_wavs/1221-135766-
↪0001.wav \
  ./icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/test_wavs/1221-135766-
↪0002.wav
```

The output is:

```
2021-11-10 14:00:55.279 [I] k2/torch/bin/ngram_lm_rescore.cu:122:int main(int, char**)␣
↪Use GPU
2021-11-10 14:00:55.280 [I] k2/torch/bin/ngram_lm_rescore.cu:126:int main(int, char**)␣
↪Device: cuda:0
2021-11-10 14:00:55.280 [I] k2/torch/bin/ngram_lm_rescore.cu:135:int main(int, char**)␣
↪Load wave files
2021-11-10 14:01:01.214 [I] k2/torch/bin/ngram_lm_rescore.cu:142:int main(int, char**)␣
↪Build Fbank computer
2021-11-10 14:01:01.215 [I] k2/torch/bin/ngram_lm_rescore.cu:153:int main(int, char**)␣
↪Compute features
2021-11-10 14:01:01.219 [I] k2/torch/bin/ngram_lm_rescore.cu:161:int main(int, char**)␣
↪Load neural network model
2021-11-10 14:01:01.945 [I] k2/torch/bin/ngram_lm_rescore.cu:176:int main(int, char**)␣
↪Compute nnet_output
2021-11-10 14:01:02.475 [I] k2/torch/bin/ngram_lm_rescore.cu:191:int main(int, char**)␣
↪Load ./icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/data/lang_bpe_500/
↪HLG.pt
2021-11-10 14:01:03.398 [I] k2/torch/bin/ngram_lm_rescore.cu:199:int main(int, char**)␣
↪Decoding
2021-11-10 14:01:03.515 [I] k2/torch/bin/ngram_lm_rescore.cu:205:int main(int, char**)␣
↪Load n-gram LM: ./icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/data/lm/
↪G_4_gram.pt
2021-11-10 14:01:07.432 [W] k2/torch/csrc/deserialization.cu:441:k2::FsaClass␣
↪k2::LoadFsa(const string&, c10::optional<c10::Device>)
Ignore non tensor attribute: 'dummy' of type: Int
2021-11-10 14:01:07.589 [I] k2/torch/bin/ngram_lm_rescore.cu:214:int main(int, char**)␣
↪Rescore with an n-gram LM
2021-11-10 14:01:08.68 [I] k2/torch/bin/ngram_lm_rescore.cu:242:int main(int, char**)
Decoding result:

./icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/test_wavs/1089-134686-
```

(continues on next page)

(continued from previous page)

```
↪0001.wav
AFTER EARLY NIGHTFALL THE YELLOW LAMPS WOULD LIGHT UP HERE AND THERE THE SQUALID QUARTER␣
↪OF THE BROTHELS

./icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/test_wavs/1221-135766-
↪0001.wav
GOD AS A DIRECT CONSEQUENCE OF THE SIN WHICH MAN THUS PUNISHED HAD GIVEN HER A LOVELY␣
↪CHILD WHOSE PLACE WAS ON THAT SAME DISHONORED BOSOM TO CONNECT HER PARENT FOREVER WITH␣
↪THE RACE AND DESCENT OF MORTALS AND TO BE FINALLY A BLESSED SOUL IN HEAVEN

./icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/test_wavs/1221-135766-
↪0002.wav
YET THESE THOUGHTS AFFECTED HESTER PRYNNE LESS WITH HOPE THAN APPREHENSION
```

HLG decoding + n-gram LM rescoring + attention decoder rescoring

```
./bin/attention_rescore \
  --use_gpu true \
  --nn_model ./icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/exp/cpu_jit.
↪pt \
  --hlg ./icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/data/lang_bpe_500/
↪HLG.pt \
  --g ./icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/data/lm/G_4_gram.pt␣
↪\
  --ngram_lm_scale 2.0 \
  --attention_scale 2.0 \
  --num_paths 100 \
  --nbest_scale 0.5 \
  --word_table ./icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/data/lang_
↪bpe_500/words.txt \
  --sos_id 1 \
  --eos_id 1 \
  ./icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/test_wavs/1089-134686-
↪0001.wav \
  ./icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/test_wavs/1221-135766-
↪0001.wav \
  ./icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/test_wavs/1221-135766-
↪0002.wav
```

The output is:

```
2021-11-10 14:02:43.656 [I] k2/torch/bin/attention_rescore.cu:149:int main(int, char**)␣
↪Use GPU
2021-11-10 14:02:43.656 [I] k2/torch/bin/attention_rescore.cu:153:int main(int, char**)␣
↪Device: cuda:0
2021-11-10 14:02:43.656 [I] k2/torch/bin/attention_rescore.cu:162:int main(int, char**)␣
↪Load wave files
2021-11-10 14:02:49.216 [I] k2/torch/bin/attention_rescore.cu:169:int main(int, char**)␣
↪Build Fbank computer
2021-11-10 14:02:49.217 [I] k2/torch/bin/attention_rescore.cu:180:int main(int, char**)␣
```

(continues on next page)

(continued from previous page)

```
↪Compute features
2021-11-10 14:02:49.222 [I] k2/torch/bin/attention_rescore.cu:188:int main(int, char**)␣
↪Load neural network model
2021-11-10 14:02:49.984 [I] k2/torch/bin/attention_rescore.cu:203:int main(int, char**)␣
↪Compute nnet_output
2021-11-10 14:02:50.624 [I] k2/torch/bin/attention_rescore.cu:220:int main(int, char**)␣
↪Load ./icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/data/lang_bpe_500/
↪HLG.pt
2021-11-10 14:02:51.519 [I] k2/torch/bin/attention_rescore.cu:228:int main(int, char**)␣
↪Decoding
2021-11-10 14:02:51.632 [I] k2/torch/bin/attention_rescore.cu:234:int main(int, char**)␣
↪Load n-gram LM: ./icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/data/lm/
↪G_4_gram.pt
2021-11-10 14:02:55.537 [W] k2/torch/csrc/deserialization.cu:441:k2::FsaClass␣
↪k2::LoadFsa(const string&, c10::optional<c10::Device>) Ignore non tensor attribute:
↪'dummy' of type: Int
2021-11-10 14:02:55.645 [I] k2/torch/bin/attention_rescore.cu:243:int main(int, char**)␣
↪Rescore with an n-gram LM
2021-11-10 14:02:55.970 [I] k2/torch/bin/attention_rescore.cu:246:int main(int, char**)␣
↪Sample 100 paths
2021-11-10 14:02:56.215 [I] k2/torch/bin/attention_rescore.cu:293:int main(int, char**)␣
↪Run attention decoder
2021-11-10 14:02:57.35 [I] k2/torch/bin/attention_rescore.cu:303:int main(int, char**)␣
↪Rescoring
2021-11-10 14:02:57.179 [I] k2/torch/bin/attention_rescore.cu:369:int main(int, char**)
Decoding result:

./icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/test_wavs/1089-134686-
↪0001.wav
AFTER EARLY NIGHTFALL THE YELLOW LAMPS WOULD LIGHT UP HERE AND THERE THE SQUALID QUARTER␣
↪OF THE BROTHELS

./icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/test_wavs/1221-135766-
↪0001.wav
GOD AS A DIRECT CONSEQUENCE OF THE SIN WHICH MAN THUS PUNISHED HAD GIVEN HER A LOVELY␣
↪CHILD WHOSE PLACE WAS ON THAT SAME DISHONORED BOSOM TO CONNECT HER PARENT FOREVER WITH␣
↪THE RACE AND DESCENT OF MORTALS AND TO BE FINALLY A BLESSED SOUL IN HEAVEN

./icefall-asr-librispeech-conformer-ctc-jit-bpe-500-2021-11-09/test_wavs/1221-135766-
↪0002.wav
YET THESE THOUGHTS AFFECTED HESTER PRYNNE LESS WITH HOPE THAN APPREHENSION
```

There is a Colab notebook showing you how to run a torch scripted model in C++. Please see

Pruned transducer statelessX

This tutorial shows you how to run a conformer transducer model with the LibriSpeech dataset.

Note: The tutorial is suitable for pruned_transducer_stateless, pruned_transducer_stateless2, pruned_transducer_stateless4, pruned_transducer_stateless5, We will take pruned_transducer_stateless4 as an example in this tutorial.

Hint: We assume you have read the page *Installation* and have setup the environment for `icefall`.

Hint: We recommend you to use a GPU or several GPUs to run this recipe.

Hint: Please scroll down to the bottom of this page to find download links for pretrained models if you don't want to train a model from scratch.

We use pruned RNN-T to compute the loss.

Note: You can find the paper about pruned RNN-T at the following address:

https://arxiv.org/abs/2206.13236

The transducer model consists of 3 parts:

- Encoder, a.k.a, the transcription network. We use a Conformer model (the reworked version by Daniel Povey)
- Decoder, a.k.a, the prediction network. We use a stateless model consisting of `nn.Embedding` and `nn.Conv1d`
- Joiner, a.k.a, the joint network.

Caution: Contrary to the conventional RNN-T models, we use a stateless decoder. That is, it has no recurrent connections.

Data preparation

Hint: The data preparation is the same as other recipes on LibriSpeech dataset, if you have finished this step, you can skip to `Training` directly.

```
$ cd egs/librispeech/ASR
$ ./prepare.sh
```

The script `./prepare.sh` handles the data preparation for you, **automagically**. All you need to do is to run it.

The data preparation contains several stages, you can use the following two options:

- `--stage`
- `--stop-stage`

to control which stage(s) should be run. By default, all stages are executed.

For example,

```
$ cd egs/librispeech/ASR
$ ./prepare.sh --stage 0 --stop-stage 0
```

means to run only stage 0.

To run stage 2 to stage 5, use:

```
$ ./prepare.sh --stage 2 --stop-stage 5
```

Hint: If you have pre-downloaded the LibriSpeech dataset and the musan dataset, say, they are saved in `/tmp/LibriSpeech` and `/tmp/musan`, you can modify the `dl_dir` variable in `./prepare.sh` to point to `/tmp` so that `./prepare.sh` won't re-download them.

Note: All generated files by `./prepare.sh`, e.g., features, lexicon, etc, are saved in `./data` directory.

We provide the following YouTube video showing how to run `./prepare.sh`.

Note: To get the latest news of next-gen Kaldi, please subscribe the following YouTube channel by Nadira Povey:

https://www.youtube.com/channel/UC_VaumpkmINz1pNkFXAN9mw

https://youtu.be/ofEIoJL-mGM

Training

Configurable options

```
$ cd egs/librispeech/ASR
$ ./pruned_transducer_stateless4/train.py --help
```

shows you the training options that can be passed from the commandline. The following options are used quite often:

- `--exp-dir`

  The directory to save checkpoints, training logs and tensorboard.

- `--full-libri`

  If it's True, the training part uses all the training data, i.e., 960 hours. Otherwise, the training part uses only the subset `train-clean-100`, which has 100 hours of training data.

  Caution: The training set is perturbed by speed with two factors: 0.9 and 1.1. If `--full-libri` is True, each epoch actually processes `3x960 == 2880` hours of data.

- `--num-epochs`

It is the number of epochs to train. For instance, `./pruned_transducer_stateless4/train.py --num-epochs 30` trains for 30 epochs and generates `epoch-1.pt`, `epoch-2.pt`, . . ., `epoch-30.pt` in the folder `./pruned_transducer_stateless4/exp`.

- `--start-epoch`

  It's used to resume training. `./pruned_transducer_stateless4/train.py --start-epoch 10` loads the checkpoint `./pruned_transducer_stateless4/exp/epoch-9.pt` and starts training from epoch 10, based on the state from epoch 9.

- `--world-size`

  It is used for multi-GPU single-machine DDP training.

  - (a) If it is 1, then no DDP training is used.
  - (b) If it is 2, then GPU 0 and GPU 1 are used for DDP training.

  The following shows some use cases with it.

  Use case 1: You have 4 GPUs, but you only want to use GPU 0 and GPU 2 for training. You can do the following:

  ```
  $ cd egs/librispeech/ASR
  $ export CUDA_VISIBLE_DEVICES="0,2"
  $ ./pruned_transducer_stateless4/train.py --world-size 2
  ```

  Use case 2: You have 4 GPUs and you want to use all of them for training. You can do the following:

  ```
  $ cd egs/librispeech/ASR
  $ ./pruned_transducer_stateless4/train.py --world-size 4
  ```

  Use case 3: You have 4 GPUs but you only want to use GPU 3 for training. You can do the following:

  ```
  $ cd egs/librispeech/ASR
  $ export CUDA_VISIBLE_DEVICES="3"
  $ ./pruned_transducer_stateless4/train.py --world-size 1
  ```

Caution: Only multi-GPU single-machine DDP training is implemented at present. Multi-GPU multi-machine DDP training will be added later.

- `--max-duration`

  It specifies the number of seconds over all utterances in a batch, before **padding**. If you encounter CUDA OOM, please reduce it.

Hint: Due to padding, the number of seconds of all utterances in a batch will usually be larger than `--max-duration`.

A larger value for `--max-duration` may cause OOM during training, while a smaller value may increase the training time. You have to tune it.

- `--use-fp16`

  If it is True, the model will train with half precision, from our experiment results, by using half precision you can train with two times larger `--max-duration` so as to get almost 2X speed up.

Pre-configured options

There are some training options, e.g., number of encoder layers, encoder dimension, decoder dimension, number of warmup steps etc, that are not passed from the commandline. They are pre-configured by the function `get_params()` in pruned_transducer_stateless4/train.py

You don't need to change these pre-configured parameters. If you really need to change them, please modify `./pruned_transducer_stateless4/train.py` directly.

Note: The options for pruned_transducer_stateless5 are a little different from other recipes. It allows you to configure `--num-encoder-layers`, `--dim-feedforward`, `--nhead`, `--encoder-dim`, `--decoder-dim`, `--joiner-dim` from commandline, so that you can train models with different size with pruned_transducer_stateless5.

Training logs

Training logs and checkpoints are saved in `--exp-dir` (e.g. `pruned_transducer_stateless4/exp`. You will find the following files in that directory:

- `epoch-1.pt`, `epoch-2.pt`, …

  These are checkpoint files saved at the end of each epoch, containing model `state_dict` and optimizer `state_dict`. To resume training from some checkpoint, say `epoch-10.pt`, you can use:

  ```
  $ ./pruned_transducer_stateless4/train.py --start-epoch 11
  ```

- `checkpoint-436000.pt`, `checkpoint-438000.pt`, …

  These are checkpoint files saved every `--save-every-n` batches, containing model `state_dict` and optimizer `state_dict`. To resume training from some checkpoint, say `checkpoint-436000`, you can use:

  ```
  $ ./pruned_transducer_stateless4/train.py --start-batch 436000
  ```

- `tensorboard/`

  This folder contains tensorBoard logs. Training loss, validation loss, learning rate, etc, are recorded in these logs. You can visualize them by:

  ```
  $ cd pruned_transducer_stateless4/exp/tensorboard
  $ tensorboard dev upload --logdir . --description "pruned␣
  ↪transducer training for LibriSpeech with icefall"
  ```

  It will print something like below:

  ```
  TensorFlow installation not found - running with reduced feature␣
  ↪set.
  Upload started and will continue reading any new data as it's added␣
  ↪to the logdir.

  To stop uploading, press Ctrl-C.

  New experiment created. View your TensorBoard at: https://
  ↪tensorboard.dev/experiment/QOGSPBgsR8KzcRMmie9JGw/

  ```

(continues on next page)

(continued from previous page)

```
[2022-11-20T15:50:50] Started scanning logdir.
Uploading 4468 scalars...
[2022-11-20T15:53:02] Total uploaded: 210171 scalars, 0 tensors, 0␣
↪binary objects
Listening for new data in logdir...
```

Note there is a URL in the above output. Click it and you will see the following screenshot:

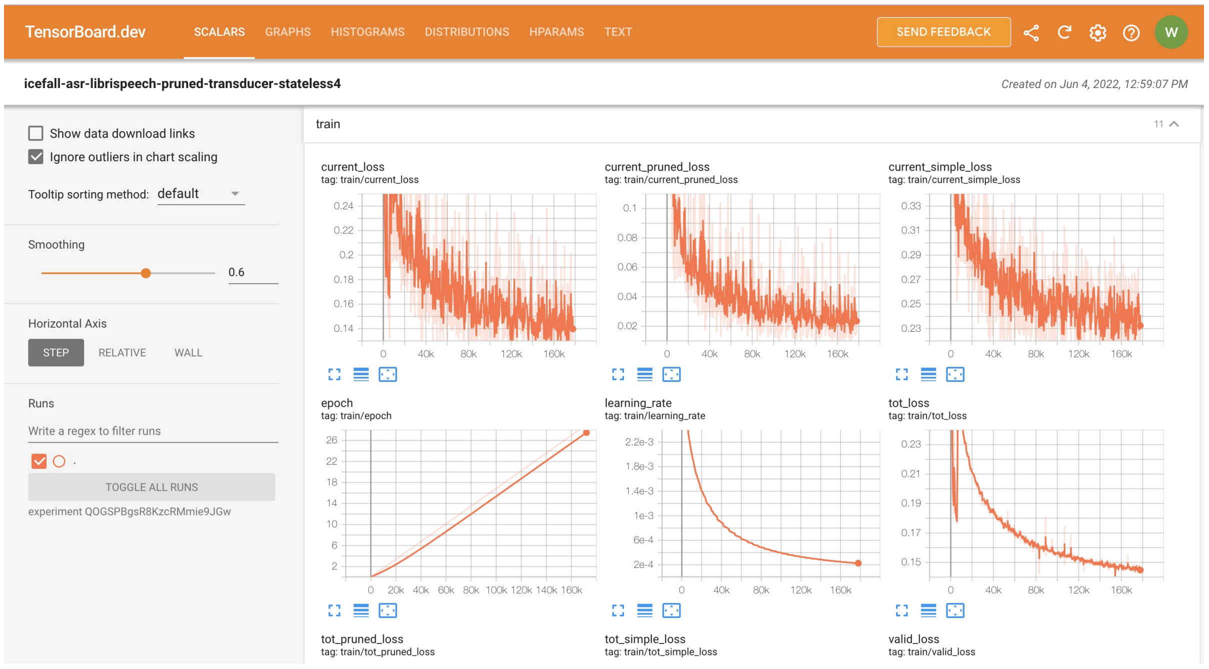


Fig. 6.5: TensorBoard screenshot.

Hint: If you don't have access to google, you can use the following command to view the tensorboard log locally:

```
cd pruned_transducer_stateless4/exp/tensorboard
tensorboard --logdir . --port 6008
```

It will print the following message:

```
Serving TensorBoard on localhost; to expose to the network, use a␣
↪proxy or pass --bind_all
TensorBoard 2.8.0 at http://localhost:6008/ (Press CTRL+C to quit)
```

Now start your browser and go to http://localhost:6008 to view the tensorboard logs.

- `log/log-train-xxxx`

  It is the detailed training log in text format, same as the one you saw printed to the console during training.

Usage example

You can use the following command to start the training using 6 GPUs:

```
export CUDA_VISIBLE_DEVICES="0,1,2,3,4,5"
./pruned_transducer_stateless4/train.py \
   --world-size 6 \
   --num-epochs 30 \
   --start-epoch 1 \
   --exp-dir pruned_transducer_stateless4/exp \
   --full-libri 1 \
   --max-duration 300
```

Decoding

The decoding part uses checkpoints saved by the training part, so you have to run the training part first.

Hint: There are two kinds of checkpoints:

- (1) `epoch-1.pt`, `epoch-2.pt`, ..., which are saved at the end of each epoch. You can pass `--epoch` to `pruned_transducer_stateless4/decode.py` to use them.
- (2) `checkpoints-436000.pt`, `epoch-438000.pt`, ..., which are saved every `--save-every-n` batches. You can pass `--iter` to `pruned_transducer_stateless4/decode.py` to use them.

We suggest that you try both types of checkpoints and choose the one that produces the lowest WERs.

```
$ cd egs/librispeech/ASR
$ ./pruned_transducer_stateless4/decode.py --help
```

shows the options for decoding.

The following shows two examples (for two types of checkpoints):

```
for m in greedy_search fast_beam_search modified_beam_search; do
  for epoch in 25 20; do
    for avg in 7 5 3 1; do
      ./pruned_transducer_stateless4/decode.py \
        --epoch $epoch \
        --avg $avg \
        --exp-dir pruned_transducer_stateless4/exp \
        --max-duration 600 \
        --decoding-method $m
    done
  done
done
```

```
for m in greedy_search fast_beam_search modified_beam_search; do
  for iter in 474000; do
    for avg in 8 10 12 14 16 18; do
      ./pruned_transducer_stateless4/decode.py \
        --iter $iter \
```

(continues on next page)

(continued from previous page)

```
        --avg $avg \
        --exp-dir pruned_transducer_stateless4/exp \
        --max-duration 600 \
        --decoding-method $m
    done
  done
done
```

Note: Supporting decoding methods are as follows:

- `greedy_search` : It takes the symbol with largest posterior probability of each frame as the decoding result.
- `beam_search` : It implements Algorithm 1 in https://arxiv.org/pdf/1211.3711.pdf and espnet/nets/beam_search_transducer.py is used as a reference. Basicly, it keeps topk states for each frame, and expands the kept states with their own contexts to next frame.
- `modified_beam_search` : It implements the same algorithm as `beam_search` above, but it runs in batch mode with `--max-sym-per-frame=1` being hardcoded.
- `fast_beam_search` : It implements graph composition between the output `log_probs` and given `FSAs`. It is hard to describe the details in several lines of texts, you can read our paper in https://arxiv.org/pdf/2211.00484.pdf or our rnnt decode code in k2. `fast_beam_search` can decode with `FSAs` on GPU efficiently.
- `fast_beam_search_LG` : The same as `fast_beam_search` above, `fast_beam_search` uses an trivial graph that has only one state, while `fast_beam_search_LG` uses an LG graph (with N-gram LM).
- `fast_beam_search_nbest` : It produces the decoding results as follows:
  - (1) Use `fast_beam_search` to get a lattice
  - (2) Select `num_paths` paths from the lattice using `k2.random_paths()`
  - (3) Unique the selected paths
  - (4) Intersect the selected paths with the lattice and compute the shortest path from the intersection result
  - (5) The path with the largest score is used as the decoding output.
- `fast_beam_search_nbest_LG` : It implements same logic as `fast_beam_search_nbest`, the only difference is that it uses `fast_beam_search_LG` to generate the lattice.

Export Model

pruned_transducer_stateless4/export.py supports exporting checkpoints from `pruned_transducer_stateless4/exp` in the following ways.

Export `model.state_dict()`

Checkpoints saved by `pruned_transducer_stateless4/train.py` also include `optimizer.state_dict()`. It is useful for resuming training. But after training, we are interested only in `model.state_dict()`. You can use the following command to extract `model.state_dict()`.

```
# Assume that --epoch 25 --avg 3 produces the smallest WER
# (You can get such information after running ./pruned_transducer_stateless4/decode.py)

epoch=25
avg=3

./pruned_transducer_stateless4/export.py \
  --exp-dir ./pruned_transducer_stateless4/exp \
  --bpe-model data/lang_bpe_500/bpe.model \
  --epoch $epoch \
  --avg  $avg
```

It will generate a file `./pruned_transducer_stateless4/exp/pretrained.pt`.

Hint: To use the generated `pretrained.pt` for `pruned_transducer_stateless4/decode.py`, you can run:

```
cd pruned_transducer_stateless4/exp
ln -s pretrained.pt epoch-999.pt
```

And then pass `--epoch 999 --avg 1 --use-averaged-model 0` to `./pruned_transducer_stateless4/decode.py`.

To use the exported model with `./pruned_transducer_stateless4/pretrained.py`, you can run:

```
./pruned_transducer_stateless4/pretrained.py \
  --checkpoint ./pruned_transducer_stateless4/exp/pretrained.pt \
  --bpe-model ./data/lang_bpe_500/bpe.model \
  --method greedy_search \
  /path/to/foo.wav \
  /path/to/bar.wav
```

Export model using `torch.jit.script()`

```
./pruned_transducer_stateless4/export.py \
  --exp-dir ./pruned_transducer_stateless4/exp \
  --bpe-model data/lang_bpe_500/bpe.model \
  --epoch 25 \
  --avg 3 \
  --jit 1
```

It will generate a file `cpu_jit.pt` in the given `exp_dir`. You can later load it by `torch.jit.load("cpu_jit.pt")`.

Note `cpu` in the name `cpu_jit.pt` means the parameters when loaded into Python are on CPU. You can use `to("cuda")` to move them to a CUDA device.

Note: You will need this `cpu_jit.pt` when deploying with Sherpa framework.

Download pretrained models

If you don't want to train from scratch, you can download the pretrained models by visiting the following links:

- pruned_transducer_stateless
- pruned_transducer_stateless2
- pruned_transducer_stateless4
- pruned_transducer_stateless5

See https://github.com/k2-fsa/icefall/blob/master/egs/librispeech/ASR/RESULTS.md for the details of the above pretrained models

Deploy with Sherpa

Please see https://k2-fsa.github.io/sherpa/python/offline_asr/conformer/librispeech.html# for how to deploy the models in `sherpa`.

Zipformer MMI

Hint: Please scroll down to the bottom of this page to find download links for pretrained models if you don't want to train a model from scratch.

This tutorial shows you how to train an Zipformer MMI model with the LibriSpeech dataset.

We use LF-MMI to compute the loss.

Note: You can find the document about LF-MMI training at the following address:

https://github.com/k2-fsa/next-gen-kaldi-wechat/blob/master/pdf/LF-MMI-training-and-decoding-in-k2-Part-I.pdf

Data preparation

```
$ cd egs/librispeech/ASR
$ ./prepare.sh
```

The script `./prepare.sh` handles the data preparation for you, **automagically**. All you need to do is to run it.

Note: We encourage you to read `./prepare.sh`.

The data preparation contains several stages. You can use the following two options:

- `--stage`

- `--stop-stage`

to control which stage(s) should be run. By default, all stages are executed.

For example,

```
$ cd egs/librispeech/ASR
$ ./prepare.sh --stage 0 --stop-stage 0
```

means to run only stage 0.

To run stage 2 to stage 5, use:

```
$ ./prepare.sh --stage 2 --stop-stage 5
```

Hint: If you have pre-downloaded the LibriSpeech dataset and the musan dataset, say, they are saved in `/tmp/LibriSpeech` and `/tmp/musan`, you can modify the `dl_dir` variable in `./prepare.sh` to point to `/tmp` so that `./prepare.sh` won't re-download them.

Note: All generated files by `./prepare.sh`, e.g., features, lexicon, etc, are saved in `./data` directory.

We provide the following YouTube video showing how to run `./prepare.sh`.

Note: To get the latest news of next-gen Kaldi, please subscribe the following YouTube channel by Nadira Povey:

https://www.youtube.com/channel/UC_VaumpkmINz1pNkFXAN9mw

https://youtu.be/ofEIoJL-mGM

Training

For stability, it uses CTC loss for model warm-up and then switches to MMI loss.

Configurable options

```
$ cd egs/librispeech/ASR
$ ./zipformer_mmi/train.py --help
```

shows you the training options that can be passed from the commandline. The following options are used quite often:

- `--full-libri`

  If it's True, the training part uses all the training data, i.e., 960 hours. Otherwise, the training part uses only the subset `train-clean-100`, which has 100 hours of training data.

  Caution: The training set is perturbed by speed with two factors: 0.9 and 1.1. If `--full-libri` is True, each epoch actually processes `3x960 == 2880` hours of data.

- `--num-epochs`

  It is the number of epochs to train. For instance, `./zipformer_mmi/train.py --num-epochs 30` trains for 30 epochs and generates `epoch-1.pt`, `epoch-2.pt`, . . . , `epoch-30.pt` in the folder `./zipformer_mmi/exp`.

- `--start-epoch`

  It's used to resume training. `./zipformer_mmi/train.py --start-epoch 10` loads the checkpoint `./zipformer_mmi/exp/epoch-9.pt` and starts training from epoch 10, based on the state from epoch 9.

- `--world-size`

  It is used for multi-GPU single-machine DDP training.

  - (a) If it is 1, then no DDP training is used.
  - (b) If it is 2, then GPU 0 and GPU 1 are used for DDP training.

  The following shows some use cases with it.

  Use case 1: You have 4 GPUs, but you only want to use GPU 0 and GPU 2 for training. You can do the following:

  ```
  $ cd egs/librispeech/ASR
  $ export CUDA_VISIBLE_DEVICES="0,2"
  $ ./zipformer_mmi/train.py --world-size 2
  ```

  Use case 2: You have 4 GPUs and you want to use all of them for training. You can do the following:

  ```
  $ cd egs/librispeech/ASR
  $ ./zipformer_mmi/train.py --world-size 4
  ```

  Use case 3: You have 4 GPUs but you only want to use GPU 3 for training. You can do the following:

  ```
  $ cd egs/librispeech/ASR
  $ export CUDA_VISIBLE_DEVICES="3"
  $ ./zipformer_mmi/train.py --world-size 1
  ```

  Caution: Only multi-GPU single-machine DDP training is implemented at present. Multi-GPU multi-machine DDP training will be added later.

- `--max-duration`

  It specifies the number of seconds over all utterances in a batch, before **padding**. If you encounter CUDA OOM, please reduce it.

  Hint: Due to padding, the number of seconds of all utterances in a batch will usually be larger than `--max-duration`.

  A larger value for `--max-duration` may cause OOM during training, while a smaller value may increase the training time. You have to tune it.

Pre-configured options

There are some training options, e.g., weight decay, number of warmup steps, results dir, etc, that are not passed from the commandline. They are pre-configured by the function `get_params()` in zipformer_mmi/train.py

You don't need to change these pre-configured parameters. If you really need to change them, please modify `./zipformer_mmi/train.py` directly.

Training logs

Training logs and checkpoints are saved in `zipformer_mmi/exp`. You will find the following files in that directory:

- `epoch-1.pt`, `epoch-2.pt`, …

  These are checkpoint files saved at the end of each epoch, containing model `state_dict` and optimizer `state_dict`. To resume training from some checkpoint, say `epoch-10.pt`, you can use:

  ```
  $ ./zipformer_mmi/train.py --start-epoch 11
  ```

- `checkpoint-436000.pt`, `checkpoint-438000.pt`, …

  These are checkpoint files saved every `--save-every-n` batches, containing model `state_dict` and optimizer `state_dict`. To resume training from some checkpoint, say `checkpoint-436000`, you can use:

  ```
  $ ./zipformer_mmi/train.py --start-batch 436000
  ```

- `tensorboard/`

  This folder contains tensorBoard logs. Training loss, validation loss, learning rate, etc, are recorded in these logs. You can visualize them by:

  ```
  $ cd zipformer_mmi/exp/tensorboard
  $ tensorboard dev upload --logdir . --description "Zipformer MMI␣
  ↪training for LibriSpeech with icefall"
  ```

  It will print something like below:

  ```
  TensorFlow installation not found - running with reduced feature␣
  ↪set.
  Upload started and will continue reading any new data as it's added␣
  ↪to the logdir.

  To stop uploading, press Ctrl-C.

  New experiment created. View your TensorBoard at: https://
  ↪tensorboard.dev/experiment/xyOZUKpEQm62HBIlUD4uPA/
  ```

  Note there is a URL in the above output. Click it and you will see tensorboard.

Hint: If you don't have access to google, you can use the following command to view the tensorboard log locally:

```
cd zipformer_mmi/exp/tensorboard
tensorboard --logdir . --port 6008
```

It will print the following message:

```
Serving TensorBoard on localhost; to expose to the network, use a␣
↪proxy or pass --bind_all
TensorBoard 2.8.0 at http://localhost:6008/ (Press CTRL+C to quit)
```

Now start your browser and go to http://localhost:6008 to view the tensorboard logs.

- `log/log-train-xxxx`

  It is the detailed training log in text format, same as the one you saw printed to the console during training.

Usage example

You can use the following command to start the training using 4 GPUs:

```
export CUDA_VISIBLE_DEVICES="0,1,2,3"
./zipformer_mmi/train.py \
  --world-size 4 \
  --num-epochs 30 \
  --start-epoch 1 \
  --full-libri 1 \
  --exp-dir zipformer_mmi/exp \
  --max-duration 500 \
  --use-fp16 1 \
  --num-workers 2
```

Decoding

The decoding part uses checkpoints saved by the training part, so you have to run the training part first.

Hint: There are two kinds of checkpoints:

- (1) `epoch-1.pt`, `epoch-2.pt`, . . ., which are saved at the end of each epoch. You can pass `--epoch` to `zipformer_mmi/decode.py` to use them.
- (2) `checkpoints-436000.pt`, `epoch-438000.pt`, . . ., which are saved every `--save-every-n` batches. You can pass `--iter` to `zipformer_mmi/decode.py` to use them.

We suggest that you try both types of checkpoints and choose the one that produces the lowest WERs.

```
$ cd egs/librispeech/ASR
$ ./zipformer_mmi/decode.py --help
```

shows the options for decoding.

The following shows the example using `epoch-*.pt`:

```
for m in nbest nbest-rescoring-LG nbest-rescoring-3-gram nbest-rescoring-4-gram; do
  ./zipformer_mmi/decode.py \
    --epoch 30 \
```

(continues on next page)

(continued from previous page)

```
    --avg 10 \
    --exp-dir ./zipformer_mmi/exp/ \
    --max-duration 100 \
    --lang-dir data/lang_bpe_500 \
    --nbest-scale 1.2 \
    --hp-scale 1.0 \
    --decoding-method $m
done
```

Export models

zipformer_mmi/export.py supports exporting checkpoints from `zipformer_mmi/exp` in the following ways.

Export `model.state_dict()`

Checkpoints saved by `zipformer_mmi/train.py` also include `optimizer.state_dict()`. It is useful for resuming training. But after training, we are interested only in `model.state_dict()`. You can use the following command to extract `model.state_dict()`.

```
./zipformer_mmi/export.py \
  --exp-dir ./zipformer_mmi/exp \
  --bpe-model data/lang_bpe_500/bpe.model \
  --epoch 30 \
  --avg 9 \
  --jit 0
```

It will generate a file `./zipformer_mmi/exp/pretrained.pt`.

Hint: To use the generated `pretrained.pt` for `zipformer_mmi/decode.py`, you can run:

```
cd zipformer_mmi/exp
ln -s pretrained epoch-9999.pt
```

And then pass `--epoch 9999 --avg 1 --use-averaged-model 0` to `./zipformer_mmi/decode.py`.

To use the exported model with `./zipformer_mmi/pretrained.py`, you can run:

```
./zipformer_mmi/pretrained.py \
  --checkpoint ./zipformer_mmi/exp/pretrained.pt \
  --bpe-model ./data/lang_bpe_500/bpe.model \
  --method 1best \
  /path/to/foo.wav \
  /path/to/bar.wav
```

Export model using `torch.jit.script()`

```
./zipformer_mmi/export.py \
  --exp-dir ./zipformer_mmi/exp \
  --bpe-model data/lang_bpe_500/bpe.model \
  --epoch 30 \
  --avg 9 \
  --jit 1
```

It will generate a file `cpu_jit.pt` in the given `exp_dir`. You can later load it by `torch.jit.load("cpu_jit.pt")`.

Note `cpu` in the name `cpu_jit.pt` means the parameters when loaded into Python are on CPU. You can use `to("cuda")` to move them to a CUDA device.

To use the generated files with `./zipformer_mmi/jit_pretrained.py`:

```
./zipformer_mmi/jit_pretrained.py \
  --nn-model-filename ./zipformer_mmi/exp/cpu_jit.pt \
  --bpe-model ./data/lang_bpe_500/bpe.model \
  --method 1best \
  /path/to/foo.wav \
  /path/to/bar.wav
```

Download pretrained models

If you don't want to train from scratch, you can download the pretrained models by visiting the following links:

- https://huggingface.co/Zengwei/icefall-asr-librispeech-zipformer-mmi-2022-12-08

See https://github.com/k2-fsa/icefall/blob/master/egs/librispeech/ASR/RESULTS.md for the details of the above pretrained models

Zipformer CTC Blank Skip

Hint: Please scroll down to the bottom of this page to find download links for pretrained models if you don't want to train a model from scratch.

This tutorial shows you how to train a Zipformer model based on the guidance from a co-trained CTC model using blank skip method with the LibriSpeech dataset.

Note: We use both CTC and RNN-T loss to train. During the forward pass, the encoder output is first used to calculate the CTC posterior probability; then for each output frame, if its blank posterior is bigger than some threshold, it will be simply discarded from the encoder output. To prevent information loss, we also put a convolution module similar to the one used in conformer (referred to as "LConv") before the frame reduction.

Data preparation

```
$ cd egs/librispeech/ASR
$ ./prepare.sh
```

The script `./prepare.sh` handles the data preparation for you, **automagically**. All you need to do is to run it.

Note: We encourage you to read `./prepare.sh`.

The data preparation contains several stages. You can use the following two options:

- `--stage`
- `--stop-stage`

to control which stage(s) should be run. By default, all stages are executed.

For example,

```
$ cd egs/librispeech/ASR
$ ./prepare.sh --stage 0 --stop-stage 0
```

means to run only stage 0.

To run stage 2 to stage 5, use:

```
$ ./prepare.sh --stage 2 --stop-stage 5
```

Hint: If you have pre-downloaded the LibriSpeech dataset and the musan dataset, say, they are saved in `/tmp/LibriSpeech` and `/tmp/musan`, you can modify the `dl_dir` variable in `./prepare.sh` to point to `/tmp` so that `./prepare.sh` won't re-download them.

Note: All generated files by `./prepare.sh`, e.g., features, lexicon, etc, are saved in `./data` directory.

We provide the following YouTube video showing how to run `./prepare.sh`.

Note: To get the latest news of next-gen Kaldi, please subscribe the following YouTube channel by Nadira Povey:

https://www.youtube.com/channel/UC_VaumpkmINz1pNkFXAN9mw

https://youtu.be/ofEIoJL-mGM

Training

For stability, it doesn`t use blank skip method until model warm-up.

Configurable options

```
$ cd egs/librispeech/ASR
$ ./pruned_transducer_stateless7_ctc_bs/train.py --help
```

shows you the training options that can be passed from the commandline. The following options are used quite often:

- `--full-libri`

  If it's True, the training part uses all the training data, i.e., 960 hours. Otherwise, the training part uses only the subset `train-clean-100`, which has 100 hours of training data.

  > **Caution:** The training set is perturbed by speed with two factors: 0.9 and 1.1. If `--full-libri` is True, each epoch actually processes `3x960 == 2880` hours of data.

- `--num-epochs`

  It is the number of epochs to train. For instance, `./pruned_transducer_stateless7_ctc_bs/train.py --num-epochs 30` trains for 30 epochs and generates `epoch-1.pt`, `epoch-2.pt`, ..., `epoch-30.pt` in the folder `./pruned_transducer_stateless7_ctc_bs/exp`.

- `--start-epoch`

  It's used to resume training. `./pruned_transducer_stateless7_ctc_bs/train.py --start-epoch 10` loads the checkpoint `./pruned_transducer_stateless7_ctc_bs/exp/epoch-9.pt` and starts training from epoch 10, based on the state from epoch 9.

- `--world-size`

  It is used for multi-GPU single-machine DDP training.

  - (a) If it is 1, then no DDP training is used.
  - (b) If it is 2, then GPU 0 and GPU 1 are used for DDP training.

  The following shows some use cases with it.

  Use case 1: You have 4 GPUs, but you only want to use GPU 0 and GPU 2 for training. You can do the following:

  ```
  $ cd egs/librispeech/ASR
  $ export CUDA_VISIBLE_DEVICES="0,2"
  $ ./pruned_transducer_stateless7_ctc_bs/train.py --world-size 2
  ```

  Use case 2: You have 4 GPUs and you want to use all of them for training. You can do the following:

  ```
  $ cd egs/librispeech/ASR
  $ ./pruned_transducer_stateless7_ctc_bs/train.py --world-size 4
  ```

  Use case 3: You have 4 GPUs but you only want to use GPU 3 for training. You can do the following:

```
$ cd egs/librispeech/ASR
$ export CUDA_VISIBLE_DEVICES="3"
$ ./pruned_transducer_stateless7_ctc_bs/train.py --world-size 1
```

Caution: Only multi-GPU single-machine DDP training is implemented at present. Multi-GPU multi-machine DDP training will be added later.

- `--max-duration`

  It specifies the number of seconds over all utterances in a batch, before **padding**. If you encounter CUDA OOM, please reduce it.

  Hint: Due to padding, the number of seconds of all utterances in a batch will usually be larger than `--max-duration`.

  A larger value for `--max-duration` may cause OOM during training, while a smaller value may increase the training time. You have to tune it.

Pre-configured options

There are some training options, e.g., weight decay, number of warmup steps, results dir, etc, that are not passed from the commandline. They are pre-configured by the function `get_params()` in pruned_transducer_stateless7_ctc_bs/train.py

You don't need to change these pre-configured parameters. If you really need to change them, please modify `./pruned_transducer_stateless7_ctc_bs/train.py` directly.

Training logs

Training logs and checkpoints are saved in `pruned_transducer_stateless7_ctc_bs/exp`. You will find the following files in that directory:

- `epoch-1.pt`, `epoch-2.pt`, ...

  These are checkpoint files saved at the end of each epoch, containing model `state_dict` and optimizer `state_dict`. To resume training from some checkpoint, say `epoch-10.pt`, you can use:

  ```
  $ ./pruned_transducer_stateless7_ctc_bs/train.py --start-epoch 11
  ```

- `checkpoint-436000.pt`, `checkpoint-438000.pt`, ...

  These are checkpoint files saved every `--save-every-n` batches, containing model `state_dict` and optimizer `state_dict`. To resume training from some checkpoint, say `checkpoint-436000`, you can use:

  ```
  $ ./pruned_transducer_stateless7_ctc_bs/train.py --start-batch␣
  ↪436000
  ```

- `tensorboard/`

  This folder contains tensorBoard logs. Training loss, validation loss, learning rate, etc, are recorded in these logs. You can visualize them by:

```
$ cd pruned_transducer_stateless7_ctc_bs/exp/tensorboard
$ tensorboard dev upload --logdir . --description "Zipformer-CTC co-
↪training using blank skip for LibriSpeech with icefall"
```

It will print something like below:

```
TensorFlow installation not found - running with reduced feature␣
↪set.
Upload started and will continue reading any new data as it's added␣
↪to the logdir.

To stop uploading, press Ctrl-C.

New experiment created. View your TensorBoard at: https://
↪tensorboard.dev/experiment/xyOZUKpEQm62HBIlUD4uPA/
```

Note there is a URL in the above output. Click it and you will see tensorboard.

Hint: If you don't have access to google, you can use the following command to view the tensorboard log locally:

```
cd pruned_transducer_stateless7_ctc_bs/exp/tensorboard
tensorboard --logdir . --port 6008
```

It will print the following message:

```
Serving TensorBoard on localhost; to expose to the network, use a␣
↪proxy or pass --bind_all
TensorBoard 2.8.0 at http://localhost:6008/ (Press CTRL+C to quit)
```

Now start your browser and go to http://localhost:6008 to view the tensorboard logs.

- `log/log-train-xxxx`

  It is the detailed training log in text format, same as the one you saw printed to the console during training.

Usage example

You can use the following command to start the training using 4 GPUs:

```
export CUDA_VISIBLE_DEVICES="0,1,2,3"
./pruned_transducer_stateless7_ctc_bs/train.py \
  --world-size 4 \
  --num-epochs 30 \
  --start-epoch 1 \
  --full-libri 1 \
  --exp-dir pruned_transducer_stateless7_ctc_bs/exp \
  --max-duration 600 \
  --use-fp16 1
```

Decoding

The decoding part uses checkpoints saved by the training part, so you have to run the training part first.

Hint: There are two kinds of checkpoints:

- (1) `epoch-1.pt`, `epoch-2.pt`, . . . , which are saved at the end of each epoch. You can pass `--epoch` to `pruned_transducer_stateless7_ctc_bs/ctc_guide_decode_bs.py` to use them.
- (2) `checkpoints-436000.pt`, `epoch-438000.pt`, . . . , which are saved every `--save-every-n` batches. You can pass `--iter` to `pruned_transducer_stateless7_ctc_bs/ctc_guide_decode_bs.py` to use them.

We suggest that you try both types of checkpoints and choose the one that produces the lowest WERs.

```
$ cd egs/librispeech/ASR
$ ./pruned_transducer_stateless7_ctc_bs/ctc_guide_decode_bs.py --help
```

shows the options for decoding.

The following shows the example using `epoch-*.pt`:

```
for m in greedy_search fast_beam_search modified_beam_search; do
    ./pruned_transducer_stateless7_ctc_bs/ctc_guide_decode_bs.py \
        --epoch 30 \
        --avg 13 \
        --exp-dir pruned_transducer_stateless7_ctc_bs/exp \
        --max-duration 600 \
        --decoding-method $m
done
```

To test CTC branch, you can use the following command:

```
for m in ctc-decoding 1best; do
    ./pruned_transducer_stateless7_ctc_bs/ctc_guide_decode_bs.py \
        --epoch 30 \
        --avg 13 \
        --exp-dir pruned_transducer_stateless7_ctc_bs/exp \
        --max-duration 600 \
        --decoding-method $m
done
```

Export models

pruned_transducer_stateless7_ctc_bs/export.py supports exporting checkpoints from `pruned_transducer_stateless7_ctc_bs/exp` in the following ways.

Export model.state_dict()

Checkpoints saved by pruned_transducer_stateless7_ctc_bs/train.py also include optimizer.state_dict(). It is useful for resuming training. But after training, we are interested only in model.state_dict(). You can use the following command to extract model.state_dict().

```
./pruned_transducer_stateless7_ctc_bs/export.py \
  --exp-dir ./pruned_transducer_stateless7_ctc_bs/exp \
  --bpe-model data/lang_bpe_500/bpe.model \
  --epoch 30 \
  --avg 13 \
  --jit 0
```

It will generate a file ./pruned_transducer_stateless7_ctc_bs/exp/pretrained.pt.

Hint: To use the generated pretrained.pt for pruned_transducer_stateless7_ctc_bs/ctc_guide_decode_bs.py, you can run:

```
cd pruned_transducer_stateless7_ctc_bs/exp
ln -s pretrained epoch-9999.pt
```

And then pass --epoch 9999 --avg 1 --use-averaged-model 0 to ./pruned_transducer_stateless7_ctc_bs/ctc_guide_decode_bs.py.

To use the exported model with ./pruned_transducer_stateless7_ctc_bs/pretrained.py, you can run:

```
./pruned_transducer_stateless7_ctc_bs/pretrained.py \
  --checkpoint ./pruned_transducer_stateless7_ctc_bs/exp/pretrained.pt \
  --bpe-model ./data/lang_bpe_500/bpe.model \
  --method greedy_search \
  /path/to/foo.wav \
  /path/to/bar.wav
```

To test CTC branch using the exported model with ./pruned_transducer_stateless7_ctc_bs/pretrained_ctc.py:

```
./pruned_transducer_stateless7_ctc_bs/jit_pretrained_ctc.py \
  --checkpoint ./pruned_transducer_stateless7_ctc_bs/exp/pretrained.pt \
  --bpe-model data/lang_bpe_500/bpe.model \
  --method ctc-decoding \
  --sample-rate 16000 \
  /path/to/foo.wav \
  /path/to/bar.wav
```

Export model using `torch.jit.script()`

```
./pruned_transducer_stateless7_ctc_bs/export.py \
  --exp-dir ./pruned_transducer_stateless7_ctc_bs/exp \
  --bpe-model data/lang_bpe_500/bpe.model \
  --epoch 30 \
  --avg 13 \
  --jit 1
```

It will generate a file `cpu_jit.pt` in the given `exp_dir`. You can later load it by `torch.jit.load("cpu_jit.pt")`.

Note `cpu` in the name `cpu_jit.pt` means the parameters when loaded into Python are on CPU. You can use `to("cuda")` to move them to a CUDA device.

To use the generated files with `./pruned_transducer_stateless7_ctc_bs/jit_pretrained.py`:

```
./pruned_transducer_stateless7_ctc_bs/jit_pretrained.py \
  --nn-model-filename ./pruned_transducer_stateless7_ctc_bs/exp/cpu_jit.pt \
  /path/to/foo.wav \
  /path/to/bar.wav
```

To test CTC branch using the generated files with `./pruned_transducer_stateless7_ctc_bs/jit_pretrained_ctc.py`:

```
./pruned_transducer_stateless7_ctc_bs/jit_pretrained_ctc.py \
  --model-filename ./pruned_transducer_stateless7_ctc_bs/exp/cpu_jit.pt \
  --bpe-model data/lang_bpe_500/bpe.model \
  --method ctc-decoding \
  --sample-rate 16000 \
  /path/to/foo.wav \
  /path/to/bar.wav
```

Download pretrained models

If you don't want to train from scratch, you can download the pretrained models by visiting the following links:

- trained on LibriSpeech 100h: https://huggingface.co/yfyeung/icefall-asr-librispeech-pruned_transducer_stateless7_ctc_bs-2022-12-14
- trained on LibriSpeech 960h: https://huggingface.co/yfyeung/icefall-asr-librispeech-pruned_transducer_stateless7_ctc_bs-2023-01-29

See https://github.com/k2-fsa/icefall/blob/master/egs/librispeech/ASR/RESULTS.md for the details of the above pretrained models

Distillation with HuBERT

This tutorial shows you how to perform knowledge distillation in icefall with the LibriSpeech dataset. The distillation method used here is called "Multi Vector Quantization Knowledge Distillation" (MVQ-KD). Please have a look at our paper Predicting Multi-Codebook Vector Quantization Indexes for Knowledge Distillation for more details about MVQ-KD.

Note: This tutorial is based on recipe pruned_transducer_stateless4. Currently, we only implement MVQ-KD in this recipe. However, MVQ-KD is theoretically applicable to all recipes with only minor changes needed. Feel free to try out MVQ-KD in different recipes. If you encounter any problems, please open an issue here icefall.

Note: We assume you have read the page *Installation* and have setup the environment for icefall.

Hint: We recommend you to use a GPU or several GPUs to run this recipe.

Data preparation

We first prepare necessary training data for LibriSpeech. This is the same as in *Pruned transducer statelessX*.

Hint: The data preparation is the same as other recipes on LibriSpeech dataset, if you have finished this step, you can skip to *Codebook index preparation* directly.

```
$ cd egs/librispeech/ASR
$ ./prepare.sh
```

The script `./prepare.sh` handles the data preparation for you, **automagically**. All you need to do is to run it.

The data preparation contains several stages, you can use the following two options:

- `--stage`
- `--stop_stage`

to control which stage(s) should be run. By default, all stages are executed.

For example,

```
$ cd egs/librispeech/ASR
$ ./prepare.sh --stage 0 --stop_stage 0 # run only stage 0
$ ./prepare.sh --stage 2 --stop_stage 5 # run from stage 2 to stage 5
```

Hint: If you have pre-downloaded the LibriSpeech dataset and the musan dataset, say, they are saved in `/tmp/LibriSpeech` and `/tmp/musan`, you can modify the `dl_dir` variable in `./prepare.sh` to point to `/tmp` so that `./prepare.sh` won't re-download them.

Note: All generated files by `./prepare.sh`, e.g., features, lexicon, etc, are saved in `./data` directory.

We provide the following YouTube video showing how to run `./prepare.sh`.

Note: To get the latest news of next-gen Kaldi, please subscribe the following YouTube channel by Nadira Povey:

https://www.youtube.com/channel/UC_VaumpkmINz1pNkFXAN9mw

https://youtu.be/ofEIoJL-mGM

Codebook index preparation

Here, we prepare necessary data for MVQ-KD. This requires the generation of codebook indexes (please read our paper. if you are interested in details). In this tutorial, we use the pre-computed codebook indexes for convenience. The only thing you need to do is to run ./distillation_with_hubert.sh.

Note: There are 5 stages in total, the first and second stage will be automatically skipped when choosing to downloaded codebook indexes prepared by icefall. Of course, you can extract and compute the codebook indexes by yourself. This will require you downloading a HuBERT-XL model and it can take a while for the extraction of codebook indexes.

As usual, you can control the stages you want to run by specifying the following two options:

- `--stage`
- `--stop_stage`

For example,

```
$ cd egs/librispeech/ASR
$ ./distillation_with_hubert.sh --stage 0 --stop_stage 0 # run only stage 0
$ ./distillation_with_hubert.sh --stage 2 --stop_stage 4 # run from stage 2 to stage 5
```

Here are a few options in ./distillation_with_hubert.sh you need to know before you proceed.

- `--full_libri` If True, use full 960h data. Otherwise only `train-clean-100` will be used
- `--use_extracted_codebook` If True, the first two stages will be skipped and the codebook indexes uploaded by us will be downloaded.

Since we are using the pre-computed codebook indexes, we set `use_extracted_codebook=True`. If you want to do full LibriSpeech experiments, please set `full_libri=True`.

The following command downloads the pre-computed codebook indexes and prepares MVQ-augmented training manifests.

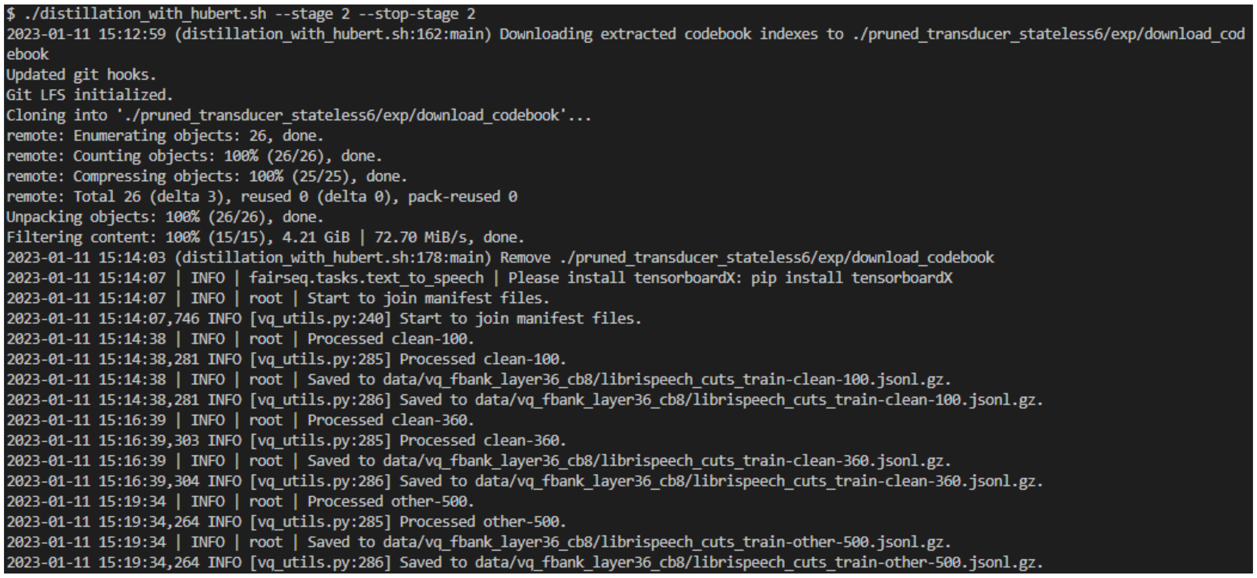
```
$ ./distillation_with_hubert.sh --stage 2 --stop_stage 2 # run only stage 2
```

Please see the following screenshot for the output of an example execution.

Hint: The codebook indexes we prepared for you in this tutorial are extracted from the 36-th layer of a fine-tuned HuBERT-XL model with 8 codebooks. If you want to try other configurations, please set `use_extracted_codebook=False` and set `embedding_layer` and `num_codebooks` by yourself.

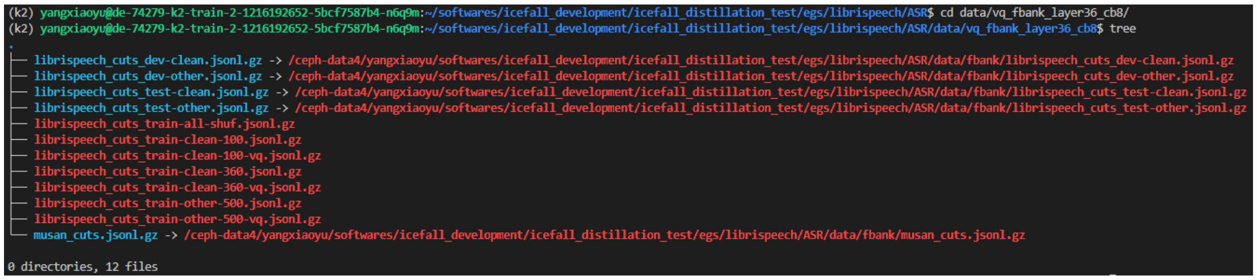
Now, you should see the following files under the directory `./data/vq_fbank_layer36_cb8`.

Whola! You are ready to perform knowledge distillation training now!



```
$ ./distillation_with_hubert.sh --stage 2 --stop-stage 2
2023-01-11 15:12:59 (distillation_with_hubert.sh:162:main) Downloading extracted codebook indexes to ./pruned_transducer_stateless6/exp/download_cod
ebook
Updated git hooks.
Git LFS initialized.
Cloning into './pruned_transducer_stateless6/exp/download_codebook'...
remote: Enumerating objects: 26, done.
remote: Counting objects: 100% (26/26), done.
remote: Compressing objects: 100% (25/25), done.
remote: Total 26 (delta 3), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (26/26), done.
Filtering content: 100% (15/15), 4.21 GiB | 72.70 MiB/s, done.
2023-01-11 15:14:03 (distillation_with_hubert.sh:178:main) Remove ./pruned_transducer_stateless6/exp/download_codebook
2023-01-11 15:14:07 | INFO | fairseq.tasks.text_to_speech | Please install tensorboardX: pip install tensorboardX
2023-01-11 15:14:07 | INFO | root | Start to join manifest files.
2023-01-11 15:14:07,746 INFO [vq_utils.py:240] Start to join manifest files.
2023-01-11 15:14:38 | INFO | root | Processed clean-100.
2023-01-11 15:14:38,281 INFO [vq_utils.py:285] Processed clean-100.
2023-01-11 15:14:38 | INFO | root | Saved to data/vq_fbank_layer36_cb8/librispeech_cuts_train-clean-100.jsonl.gz.
2023-01-11 15:14:38,281 INFO [vq_utils.py:286] Saved to data/vq_fbank_layer36_cb8/librispeech_cuts_train-clean-100.jsonl.gz.
2023-01-11 15:16:39 | INFO | root | Processed clean-360.
2023-01-11 15:16:39,303 INFO [vq_utils.py:285] Processed clean-360.
2023-01-11 15:16:39 | INFO | root | Saved to data/vq_fbank_layer36_cb8/librispeech_cuts_train-clean-360.jsonl.gz.
2023-01-11 15:16:39,304 INFO [vq_utils.py:286] Saved to data/vq_fbank_layer36_cb8/librispeech_cuts_train-clean-360.jsonl.gz.
2023-01-11 15:19:34 | INFO | root | Processed other-500.
2023-01-11 15:19:34,264 INFO [vq_utils.py:285] Processed other-500.
2023-01-11 15:19:34 | INFO | root | Saved to data/vq_fbank_layer36_cb8/librispeech_cuts_train-other-500.jsonl.gz.
2023-01-11 15:19:34,264 INFO [vq_utils.py:286] Saved to data/vq_fbank_layer36_cb8/librispeech_cuts_train-other-500.jsonl.gz.
```

Fig. 6.6: Downloading codebook indexes and preparing training manifest.



```
(k2) yangxiaoyu@de-74279-k2-train-2-1216192652-5bcf7587b4-n6q9m:~/softwares/icefall_development/icefall_distillation_test/egs/librispeech/ASR$ cd data/vq_fbank_layer36_cb8/
(k2) yangxiaoyu@de-74279-k2-train-2-1216192652-5bcf7587b4-n6q9m:~/softwares/icefall_development/icefall_distillation_test/egs/librispeech/ASR/data/vq_fbank_layer36_cb8$ tree
.
├── librispeech_cuts_dev-clean.jsonl.gz -> /ceph-data4/yangxiaoyu/softwares/icefall_development/icefall_distillation_test/egs/librispeech/ASR/data/fbank/librispeech_cuts_dev-clean.jsonl.gz
├── librispeech_cuts_dev-other.jsonl.gz -> /ceph-data4/yangxiaoyu/softwares/icefall_development/icefall_distillation_test/egs/librispeech/ASR/data/fbank/librispeech_cuts_dev-other.jsonl.gz
├── librispeech_cuts_test-clean.jsonl.gz -> /ceph-data4/yangxiaoyu/softwares/icefall_development/icefall_distillation_test/egs/librispeech/ASR/data/fbank/librispeech_cuts_test-clean.jsonl.gz
├── librispeech_cuts_test-other.jsonl.gz -> /ceph-data4/yangxiaoyu/softwares/icefall_development/icefall_distillation_test/egs/librispeech/ASR/data/fbank/librispeech_cuts_test-other.jsonl.gz
├── librispeech_cuts_train-all-shuf.jsonl.gz
├── librispeech_cuts_train-clean-100.jsonl.gz
├── librispeech_cuts_train-clean-100-vq.jsonl.gz
├── librispeech_cuts_train-clean-360.jsonl.gz
├── librispeech_cuts_train-clean-360-vq.jsonl.gz
├── librispeech_cuts_train-other-500.jsonl.gz
├── librispeech_cuts_train-other-500-vq.jsonl.gz
└── musan_cuts.jsonl.gz -> /ceph-data4/yangxiaoyu/softwares/icefall_development/icefall_distillation_test/egs/librispeech/ASR/data/fbank/musan_cuts.jsonl.gz

0 directories, 12 files
```

Fig. 6.7: MVQ-augmented training manifests.

Training

To perform training, please run stage 3 by executing the following command.

```
$ ./prepare.sh --stage 3 --stop_stage 3 # run MVQ training
```

Here is the code snippet for training:

```
WORLD_SIZE=$(echo ${CUDA_VISIBLE_DEVICES} | awk '{n=split($1, _, ","); print n}')

./pruned_transducer_stateless6/train.py \
  --manifest-dir ./data/vq_fbank_layer36_cb8 \
  --master-port 12359 \
  --full-libri $full_libri \
  --spec-aug-time-warp-factor -1 \
  --max-duration 300 \
  --world-size ${WORLD_SIZE} \
  --num-epochs 30 \
  --exp-dir $exp_dir \
  --enable-distillation True \
  --codebook-loss-scale 0.01
```

There are a few training arguments in the following training commands that should be paid attention to.

- `--enable-distillation` If True, knowledge distillation training is enabled.
- `--codebook-loss-scale` The scale of the knowledge distillation loss.
- `--manifest-dir` The path to the MVQ-augmented manifest.

Decoding

After training finished, you can test the performance on using the following command.

```
export CUDA_VISIBLE_DEVICES=0
./pruned_transducer_stateless6/train.py \
  --decoding-method "modified_beam_search" \
  --epoch 30 \
  --avg 10 \
  --max-duration 200 \
  --exp-dir $exp_dir \
  --enable-distillation True
```

You should get similar results as here.

That's all! Feel free to experiment with your own setups and report your results. If you encounter any problems during training, please open up an issue here.

6.1.3 TIMIT

TDNN-LiGRU-CTC

This tutorial shows you how to run a TDNN-LiGRU-CTC model with the TIMIT dataset.

Hint: We assume you have read the page *Installation* and have setup the environment for `icefall`.

Data preparation

```
$ cd egs/timit/ASR
$ ./prepare.sh
```

The script `./prepare.sh` handles the data preparation for you, **automagically**. All you need to do is to run it.

The data preparation contains several stages, you can use the following two options:

- `--stage`
- `--stop-stage`

to control which stage(s) should be run. By default, all stages are executed.

For example,

```
$ cd egs/timit/ASR
$ ./prepare.sh --stage 0 --stop-stage 0
```

means to run only stage 0.

To run stage 2 to stage 5, use:

```
$ ./prepare.sh --stage 2 --stop-stage 5
```

Training

Now describing the training of TDNN-LiGRU-CTC model, contained in the tdnn_ligru_ctc folder.

Hint: TIMIT is a very small dataset. So one GPU is enough.

The command to run the training part is:

```
$ cd egs/timit/ASR
$ export CUDA_VISIBLE_DEVICES="0"
$ ./tdnn_ligru_ctc/train.py
```

By default, it will run `25` epochs. Training logs and checkpoints are saved in `tdnn_ligru_ctc/exp`.

In `tdnn_ligru_ctc/exp`, you will find the following files:

- `epoch-0.pt`, `epoch-1.pt`, . . . , `epoch-29.pt`

  These are checkpoint files, containing model `state_dict` and optimizer `state_dict`. To resume training from some checkpoint, say `epoch-10.pt`, you can use:

```
$ ./tdnn_ligru_ctc/train.py --start-epoch 11
```

- `tensorboard/`

  This folder contains TensorBoard logs. Training loss, validation loss, learning rate, etc, are recorded in these logs. You can visualize them by:

  ```
  $ cd tdnn_ligru_ctc/exp/tensorboard
  $ tensorboard dev upload --logdir . --description "TDNN ligru training for␣
  ↪timit with icefall"
  ```

- `log/log-train-xxxx`

  It is the detailed training log in text format, same as the one you saw printed to the console during training.

To see available training options, you can use:

```
$ ./tdnn_ligru_ctc/train.py --help
```

Other training options, e.g., learning rate, results dir, etc., are pre-configured in the function `get_params()` in tdnn_ligru_ctc/train.py. Normally, you don't need to change them. You can change them by modifying the code, if you want.

Decoding

The decoding part uses checkpoints saved by the training part, so you have to run the training part first.

The command for decoding is:

```
$ export CUDA_VISIBLE_DEVICES="0"
$ ./tdnn_ligru_ctc/decode.py
```

You will see the WER in the output log.

Decoded results are saved in `tdnn_ligru_ctc/exp`.

```
$ ./tdnn_ligru_ctc/decode.py --help
```

shows you the available decoding options.

Some commonly used options are:

- `--epoch`

  You can select which checkpoint to be used for decoding. For instance, `./tdnn_ligru_ctc/decode.py --epoch 10` means to use `./tdnn_ligru_ctc/exp/epoch-10.pt` for decoding.

- `--avg`

  It's related to model averaging. It specifies number of checkpoints to be averaged. The averaged model is used for decoding. For example, the following command:

  ```
  $ ./tdnn_ligru_ctc/decode.py --epoch 25 --avg 17
  ```

  uses the average of `epoch-9.pt`, `epoch-10.pt`, `epoch-11.pt`, `epoch-12.pt`, `epoch-13.pt`, `epoch-14.pt`, `epoch-15.pt`, `epoch-16.pt`, `epoch-17.pt`, `epoch-18.pt`, `epoch-19.pt`, `epoch-20.pt`, `epoch-21.pt`, `epoch-22.pt`, `epoch-23.pt`, `epoch-24.pt` and `epoch-25.pt` for decoding.

- `--export`

  If it is True, i.e., `./tdnn_ligru_ctc/decode.py --export 1`, the code will save the averaged model to `tdnn_ligru_ctc/exp/pretrained.pt`. See *Pre-trained Model* for how to use it.

Pre-trained Model

We have uploaded the pre-trained model to https://huggingface.co/luomingshuang/icefall_asr_timit_tdnn_ligru_ctc.

The following shows you how to use the pre-trained model.

Install kaldifeat

kaldifeat is used to extract features for a single sound file or multiple sound files at the same time.

Please refer to https://github.com/csukuangfj/kaldifeat for installation.

Download the pre-trained model

```
$ cd egs/timit/ASR
$ mkdir tmp-ligru
$ cd tmp-ligru
$ git lfs install
$ git clone https://huggingface.co/luomingshuang/icefall_asr_timit_tdnn_ligru_ctc
```

Caution: You have to use `git lfs` to download the pre-trained model.

Caution: In order to use this pre-trained model, your k2 version has to be v1.7 or later.

After downloading, you will have the following files:

```
$ cd egs/timit/ASR
$ tree tmp-ligru
```

```
tmp-ligru/
`-- icefall_asr_timit_tdnn_ligru_ctc
    |-- README.md
    |-- data
    |   |-- lang_phone
    |   |   |-- HLG.pt
    |   |   |-- tokens.txt
    |   |   `-- words.txt
    |   `-- lm
    |       `-- G_4_gram.pt
    |-- exp
    |   `-- pretrained_average_9_25.pt
    `-- test_wavs
        |-- FDHC0_SI1559.WAV
```

(continues on next page)

(continued from previous page)

```
        |-- FELC0_SI756.WAV
        |-- FMGD0_SI1564.WAV
        `-- trans.txt

6 directories, 10 files
```

File descriptions:

- `data/lang_phone/HLG.pt`

  It is the decoding graph.

- `data/lang_phone/tokens.txt`

  It contains tokens and their IDs.

- `data/lang_phone/words.txt`

  It contains words and their IDs.

- `data/lm/G_4_gram.pt`

  It is a 4-gram LM, useful for LM rescoring.

- `exp/pretrained.pt`

  It contains pre-trained model parameters, obtained by averaging checkpoints from `epoch-9.pt` to `epoch-25.pt`. Note: We have removed optimizer `state_dict` to reduce file size.

- `test_waves/*.WAV`

  It contains some test sound files from timit TEST dataset.

- `test_waves/trans.txt`

  It contains the reference transcripts for the sound files in `test_waves/`.

The information of the test sound files is listed below:

```
$ ffprobe -show_format tmp-ligru/icefall_asr_timit_tdnn_ligru_ctc/test_waves/FDHC0_
↪SI1559.WAV

Input #0, nistsphere, from 'tmp-ligru/icefall_asr_timit_tdnn_ligru_ctc/test_waves/FDHC0_
↪SI1559.WAV':
Metadata:
  database_id     : TIMIT
  database_version: 1.0
  utterance_id    : dhc0_si1559
  sample_min      : -4176
  sample_max      : 5984
Duration: 00:00:03.40, bitrate: 258 kb/s
  Stream #0:0: Audio: pcm_s16le, 16000 Hz, 1 channels, s16, 256 kb/s

$ ffprobe -show_format tmp-ligru/icefall_asr_timit_tdnn_ligru_ctc/test_waves/FELC0_SI756.
↪WAV

Input #0, nistsphere, from 'tmp-ligru/icefall_asr_timit_tdnn_ligru_ctc/test_waves/FELC0_
↪SI756.WAV':
Metadata:
  database_id     : TIMIT
```

(continues on next page)

(continued from previous page)

```
  database_version: 1.0
  utterance_id    : elc0_si756
  sample_min      : -1546
  sample_max      : 1989
Duration: 00:00:04.19, bitrate: 257 kb/s
  Stream #0:0: Audio: pcm_s16le, 16000 Hz, 1 channels, s16, 256 kb/s

$ ffprobe -show_format tmp-ligru/icefall_asr_timit_tdnn_ligru_ctc/test_waves/FMGD0_
↪SI1564.WAV

Input #0, nistsphere, from 'tmp-ligru/icefall_asr_timit_tdnn_ligru_ctc/test_waves/FMGD0_
↪SI1564.WAV':
Metadata:
  database_id     : TIMIT
  database_version: 1.0
  utterance_id    : mgd0_si1564
  sample_min      : -7626
  sample_max      : 10573
Duration: 00:00:04.44, bitrate: 257 kb/s
  Stream #0:0: Audio: pcm_s16le, 16000 Hz, 1 channels, s16, 256 kb/s
```

Inference with a pre-trained model

```
$ cd egs/timit/ASR
$ ./tdnn_ligru_ctc/pretrained.py --help
```

shows the usage information of `./tdnn_ligru_ctc/pretrained.py`.

To decode with `1best` method, we can use:

```
./tdnn_ligru_ctc/pretrained.py
  --method 1best
  --checkpoint ./tmp-ligru/icefall_asr_timit_tdnn_ligru_ctc/exp/pretrained_average_9_25.
↪pt
  --words-file ./tmp-ligru/icefall_asr_timit_tdnn_ligru_ctc/data/lang_phone/words.txt
  --HLG ./tmp-ligru/icefall_asr_timit_tdnn_ligru_ctc/data/lang_phone/HLG.pt
  ./tmp-ligru/icefall_asr_timit_tdnn_ligru_ctc/test_waves/FDHC0_SI1559.WAV
  ./tmp-ligru/icefall_asr_timit_tdnn_ligru_ctc/test_waves/FELC0_SI756.WAV
  ./tmp-ligru/icefall_asr_timit_tdnn_ligru_ctc/test_waves/FMGD0_SI1564.WAV
```

The output is:

```
2021-11-08 20:41:33,660 INFO [pretrained.py:169] device: cuda:0
2021-11-08 20:41:33,660 INFO [pretrained.py:171] Creating model
2021-11-08 20:41:38,680 INFO [pretrained.py:183] Loading HLG from ./tmp-ligru/icefall_
↪asr_timit_tdnn_ligru_ctc/data/lang_phone/HLG.pt
2021-11-08 20:41:38,695 INFO [pretrained.py:200] Constructing Fbank computer
2021-11-08 20:41:38,697 INFO [pretrained.py:210] Reading sound files: ['./tmp-ligru/
↪icefall_asr_timit_tdnn_ligru_ctc/test_waves/FDHC0_SI1559.WAV', './tmp-ligru/icefall_
↪asr_timit_tdnn_ligru_ctc/test_waves/FELC0_SI756.WAV', './tmp-ligru/icefall_asr_timit_
↪tdnn_ligru_ctc/test_waves/FMGD0_SI1564.WAV']
```

(continues on next page)

(continued from previous page)

```
2021-11-08 20:41:38,704 INFO [pretrained.py:216] Decoding started
2021-11-08 20:41:39,819 INFO [pretrained.py:246] Use HLG decoding
2021-11-08 20:41:39,829 INFO [pretrained.py:267]
./tmp-ligru/icefall_asr_timit_tdnn_ligru_ctc/test_waves/FDHC0_SI1559.WAV:
sil dh ih sh uw ah l iy v iy z ih sil p r aa sil k s ih m ey dx ih sil d w uh dx ih w ih␣
↪s f iy l ih ng w ih th ih n ih m s eh l f sil jh

./tmp-ligru/icefall_asr_timit_tdnn_ligru_ctc/test_waves/FELC0_SI756.WAV:
sil m ih sil t ih r iy s sil s er r ih m ih sil m aa l ih sil k l ey sil r eh sil d w ay␣
↪sil d aa r sil b ah f sil jh

./tmp-ligru/icefall_asr_timit_tdnn_ligru_ctc/test_waves/FMGD0_SI1564.WAV:
sil hh ah z sil b ih sil g r iy w ah z sil d aw n ih sil b ay s sil n ey sil w eh l f eh␣
↪n s ih z eh n dh eh r w er sil g r ey z ih ng sil k ae dx l sil


2021-11-08 20:41:39,829 INFO [pretrained.py:269] Decoding Done
```

To decode with whole-lattice-rescoring methond, you can use

```
./tdnn_ligru_ctc/pretrained.py \
  --method whole-lattice-rescoring \
  --checkpoint ./tmp-ligru/icefall_asr_timit_tdnn_ligru_ctc/exp/pretrained_average_9_25.
↪pt \
  --words-file ./tmp-ligru/icefall_asr_timit_tdnn_ligru_ctc/data/lang_phone/words.txt \
  --HLG ./tmp-ligru/icefall_asr_timit_tdnn_ligru_ctc/data/lang_phone/HLG.pt \
  --G ./tmp-ligru/icefall_asr_timit_tdnn_ligru_ctc/data/lm/G_4_gram.pt \
  --ngram-lm-scale 0.1 \
  ./tmp-ligru/icefall_asr_timit_tdnn_ligru_ctc/test_waves/FDHC0_SI1559.WAV
  ./tmp-ligru/icefall_asr_timit_tdnn_ligru_ctc/test_waves/FELC0_SI756.WAV
  ./tmp-ligru/icefall_asr_timit_tdnn_ligru_ctc/test_waves/FMGD0_SI1564.WAV
```

The decoding output is:

```
2021-11-08 20:37:50,693 INFO [pretrained.py:169] device: cuda:0
2021-11-08 20:37:50,693 INFO [pretrained.py:171] Creating model
2021-11-08 20:37:54,693 INFO [pretrained.py:183] Loading HLG from ./tmp-ligru/icefall_
↪asr_timit_tdnn_ligru_ctc/data/lang_phone/HLG.pt
2021-11-08 20:37:54,705 INFO [pretrained.py:191] Loading G from ./tmp-ligru/icefall_asr_
↪timit_tdnn_ligru_ctc/data/lm/G_4_gram.pt
2021-11-08 20:37:54,714 INFO [pretrained.py:200] Constructing Fbank computer
2021-11-08 20:37:54,715 INFO [pretrained.py:210] Reading sound files: ['./tmp-ligru/
↪icefall_asr_timit_tdnn_ligru_ctc/test_waves/FDHC0_SI1559.WAV', './tmp-ligru/icefall_
↪asr_timit_tdnn_ligru_ctc/test_waves/FELC0_SI756.WAV', './tmp-ligru/icefall_asr_timit_
↪tdnn_ligru_ctc/test_waves/FMGD0_SI1564.WAV']
2021-11-08 20:37:54,720 INFO [pretrained.py:216] Decoding started
2021-11-08 20:37:55,808 INFO [pretrained.py:251] Use HLG decoding + LM rescoring
2021-11-08 20:37:56,348 INFO [pretrained.py:267]
./tmp-ligru/icefall_asr_timit_tdnn_ligru_ctc/test_waves/FDHC0_SI1559.WAV:
sil dh ih sh uw ah l iy v iy z ah sil p r aa sil k s ih m ey dx ih sil d w uh dx iy w ih␣
↪s f iy l iy ng w ih th ih n ih m s eh l f sil jh

```

(continues on next page)

(continued from previous page)

```
./tmp-ligru/icefall_asr_timit_tdnn_ligru_ctc/test_waves/FELC0_SI756.WAV:
sil m ih sil t ih r iy l s sil s er r eh m ih sil m aa l ih ng sil k l ey sil r eh sil d␣
↪w ay sil d aa r sil b ah f sil jh ch

./tmp-ligru/icefall_asr_timit_tdnn_ligru_ctc/test_waves/FMGD0_SI1564.WAV:
sil hh ah z sil b ih n sil g r iy w ah z sil b aw n ih sil b ay s sil n ey sil w er l f␣
↪eh n s ih z eh n dh eh r w er sil g r ey z ih ng sil k ae dx l sil


2021-11-08 20:37:56,348 INFO [pretrained.py:269] Decoding Done
```

Colab notebook

We provide a colab notebook for decoding with pre-trained model.

Congratulations! You have finished the TDNN-LiGRU-CTC recipe on timit in `icefall`.

TDNN-LSTM-CTC

This tutorial shows you how to run a TDNN-LSTM-CTC model with the TIMIT dataset.

Hint: We assume you have read the page *Installation* and have setup the environment for `icefall`.

Data preparation

```
$ cd egs/timit/ASR
$ ./prepare.sh
```

The script `./prepare.sh` handles the data preparation for you, **automagically**. All you need to do is to run it.

The data preparation contains several stages, you can use the following two options:

- `--stage`
- `--stop-stage`

to control which stage(s) should be run. By default, all stages are executed.

For example,

```
$ cd egs/timit/ASR
$ ./prepare.sh --stage 0 --stop-stage 0
```

means to run only stage 0.

To run stage 2 to stage 5, use:

```
$ ./prepare.sh --stage 2 --stop-stage 5
```

Training

Now describing the training of TDNN-LSTM-CTC model, contained in the tdnn_lstm_ctc folder.

Hint: TIMIT is a very small dataset. So one GPU for training is enough.

The command to run the training part is:

```
$ cd egs/timit/ASR
$ export CUDA_VISIBLE_DEVICES="0"
$ ./tdnn_lstm_ctc/train.py
```

By default, it will run `25` epochs. Training logs and checkpoints are saved in `tdnn_lstm_ctc/exp`.

In `tdnn_lstm_ctc/exp`, you will find the following files:

- `epoch-0.pt`, `epoch-1.pt`, . . . , `epoch-29.pt`

  These are checkpoint files, containing model `state_dict` and optimizer `state_dict`. To resume training from some checkpoint, say `epoch-10.pt`, you can use:

  ```
  $ ./tdnn_lstm_ctc/train.py --start-epoch 11
  ```

- `tensorboard/`

  This folder contains TensorBoard logs. Training loss, validation loss, learning rate, etc, are recorded in these logs. You can visualize them by:

  ```
  $ cd tdnn_lstm_ctc/exp/tensorboard
  $ tensorboard dev upload --logdir . --description "TDNN LSTM training for␣
  ↪timit with icefall"
  ```

- `log/log-train-xxxx`

  It is the detailed training log in text format, same as the one you saw printed to the console during training.

To see available training options, you can use:

```
$ ./tdnn_lstm_ctc/train.py --help
```

Other training options, e.g., learning rate, results dir, etc., are pre-configured in the function `get_params()` in tdnn_lstm_ctc/train.py. Normally, you don't need to change them. You can change them by modifying the code, if you want.

Decoding

The decoding part uses checkpoints saved by the training part, so you have to run the training part first.

The command for decoding is:

```
$ export CUDA_VISIBLE_DEVICES="0"
$ ./tdnn_lstm_ctc/decode.py
```

You will see the WER in the output log.

Decoded results are saved in `tdnn_lstm_ctc/exp`.

```
$ ./tdnn_lstm_ctc/decode.py --help
```

shows you the available decoding options.

Some commonly used options are:

- `--epoch`

  You can select which checkpoint to be used for decoding. For instance, `./tdnn_lstm_ctc/decode.py --epoch 10` means to use `./tdnn_lstm_ctc/exp/epoch-10.pt` for decoding.

- `--avg`

  It's related to model averaging. It specifies number of checkpoints to be averaged. The averaged model is used for decoding. For example, the following command:

  ```
  $ ./tdnn_lstm_ctc/decode.py --epoch 25 --avg 10
  ```

  uses the average of `epoch-16.pt`, `epoch-17.pt`, `epoch-18.pt`, `epoch-19.pt`, `epoch-20.pt`, `epoch-21.pt`, `epoch-22.pt`, `epoch-23.pt`, `epoch-24.pt` and `epoch-25.pt` for decoding.

- `--export`

  If it is `True`, i.e., `./tdnn_lstm_ctc/decode.py --export 1`, the code will save the averaged model to `tdnn_lstm_ctc/exp/pretrained.pt`. See *Pre-trained Model* for how to use it.

Pre-trained Model

We have uploaded the pre-trained model to https://huggingface.co/luomingshuang/icefall_asr_timit_tdnn_lstm_ctc.

The following shows you how to use the pre-trained model.

Install kaldifeat

kaldifeat is used to extract features for a single sound file or multiple sound files at the same time.

Please refer to https://github.com/csukuangfj/kaldifeat for installation.

Download the pre-trained model

```
$ cd egs/timit/ASR
$ mkdir tmp-lstm
$ cd tmp-lstm
$ git lfs install
$ git clone https://huggingface.co/luomingshuang/icefall_asr_timit_tdnn_lstm_ctc
```

Caution: You have to use `git lfs` to download the pre-trained model.

Caution: In order to use this pre-trained model, your k2 version has to be v1.7 or later.

After downloading, you will have the following files:

```
$ cd egs/timit/ASR
$ tree tmp-lstm
```

```
tmp-lstm/
`-- icefall_asr_timit_tdnn_lstm_ctc
    |-- README.md
    |-- data
    |   |-- lang_phone
    |   |   |-- HLG.pt
    |   |   |-- tokens.txt
    |   |   `-- words.txt
    |   `-- lm
    |       `-- G_4_gram.pt
    |-- exp
    |   `-- pretrained_average_16_25.pt
    `-- test_wavs
        |-- FDHC0_SI1559.WAV
        |-- FELC0_SI756.WAV
        |-- FMGD0_SI1564.WAV
        `-- trans.txt

6 directories, 10 files
```

File descriptions:

- `data/lang_phone/HLG.pt`

  It is the decoding graph.

- `data/lang_phone/tokens.txt`

  It contains tokens and their IDs.

- `data/lang_phone/words.txt`

  It contains words and their IDs.

- `data/lm/G_4_gram.pt`

  It is a 4-gram LM, useful for LM rescoring.

- `exp/pretrained.pt`

  It contains pre-trained model parameters, obtained by averaging checkpoints from `epoch-16.pt` to `epoch-25.pt`. Note: We have removed optimizer `state_dict` to reduce file size.

- `test_waves/*.WAV`

  It contains some test sound files from timit `TEST` dataset.

- `test_waves/trans.txt`

  It contains the reference transcripts for the sound files in `test_waves/`.

The information of the test sound files is listed below:

```
$ ffprobe -show_format tmp-lstm/icefall_asr_timit_tdnn_lstm_ctc/test_waves/FDHC0_SI1559.
↪WAV

Input #0, nistsphere, from 'tmp-lstm/icefall_asr_timit_tdnn_lstm_ctc/test_waves/FDHC0_
```

(continues on next page)

(continued from previous page)

```
↪SI1559.WAV':
Metadata:
  database_id     : TIMIT
  database_version: 1.0
  utterance_id    : dhc0_si1559
  sample_min      : -4176
  sample_max      : 5984
Duration: 00:00:03.40, bitrate: 258 kb/s
  Stream #0:0: Audio: pcm_s16le, 16000 Hz, 1 channels, s16, 256 kb/s

$ ffprobe -show_format tmp-lstm/icefall_asr_timit_tdnn_lstm_ctc/test_waves/FELC0_SI756.
↪WAV

Input #0, nistsphere, from 'tmp-lstm/icefall_asr_timit_tdnn_lstm_ctc/test_waves/FELC0_
↪SI756.WAV':
Metadata:
  database_id     : TIMIT
  database_version: 1.0
  utterance_id    : elc0_si756
  sample_min      : -1546
  sample_max      : 1989
Duration: 00:00:04.19, bitrate: 257 kb/s
  Stream #0:0: Audio: pcm_s16le, 16000 Hz, 1 channels, s16, 256 kb/s

$ ffprobe -show_format tmp-lstm/icefall_asr_timit_tdnn_lstm_ctc/test_waves/FMGD0_SI1564.
↪WAV

Input #0, nistsphere, from 'tmp-lstm/icefall_asr_timit_tdnn_lstm_ctc/test_waves/FMGD0_
↪SI1564.WAV':
Metadata:
  database_id     : TIMIT
  database_version: 1.0
  utterance_id    : mgd0_si1564
  sample_min      : -7626
  sample_max      : 10573
Duration: 00:00:04.44, bitrate: 257 kb/s
  Stream #0:0: Audio: pcm_s16le, 16000 Hz, 1 channels, s16, 256 kb/s
```

Inference with a pre-trained model

```
$ cd egs/timit/ASR
$ ./tdnn_lstm_ctc/pretrained.py --help
```

shows the usage information of `./tdnn_lstm_ctc/pretrained.py`.

To decode with `1best` method, we can use:

```
./tdnn_lstm_ctc/pretrained.py
  --method 1best
  --checkpoint ./tmp-lstm/icefall_asr_timit_tdnn_lstm_ctc/exp/pretrained_average_16_25.pt
  --words-file ./tmp-lstm/icefall_asr_timit_tdnn_lstm_ctc/data/lang_phone/words.txt
```

(continues on next page)

(continued from previous page)

```
  --HLG ./tmp-lstm/icefall_asr_timit_tdnn_lstm_ctc/data/lang_phone/HLG.pt
  ./tmp-lstm/icefall_asr_timit_tdnn_lstm_ctc/test_waves/FDHC0_SI1559.WAV
  ./tmp-lstm/icefall_asr_timit_tdnn_lstm_ctc/test_waves/FELC0_SI756.WAV
  ./tmp-lstm/icefall_asr_timit_tdnn_lstm_ctc/test_waves/FMGD0_SI1564.WAV
```

The output is:

```
2021-11-08 21:02:49,583 INFO [pretrained.py:169] device: cuda:0
2021-11-08 21:02:49,584 INFO [pretrained.py:171] Creating model
2021-11-08 21:02:53,816 INFO [pretrained.py:183] Loading HLG from ./tmp-lstm/icefall_asr_
↪timit_tdnn_lstm_ctc/data/lang_phone/HLG.pt
2021-11-08 21:02:53,827 INFO [pretrained.py:200] Constructing Fbank computer
2021-11-08 21:02:53,827 INFO [pretrained.py:210] Reading sound files: ['./tmp-lstm/
↪icefall_asr_timit_tdnn_lstm_ctc/test_waves/FDHC0_SI1559.WAV', './tmp-lstm/icefall_asr_
↪timit_tdnn_lstm_ctc/test_waves/FELC0_SI756.WAV', './tmp-lstm/icefall_asr_timit_tdnn_
↪lstm_ctc/test_waves/FMGD0_SI1564.WAV']
2021-11-08 21:02:53,831 INFO [pretrained.py:216] Decoding started
2021-11-08 21:02:54,380 INFO [pretrained.py:246] Use HLG decoding
2021-11-08 21:02:54,387 INFO [pretrained.py:267]
./tmp-lstm/icefall_asr_timit_tdnn_lstm_ctc/test_waves/FDHC0_SI1559.WAV:
sil dh ih sh uw ah l iy v iy z ih sil p r aa sil k s ih m ey dx ih sil d w uh dx iy w ih␣
↪s f iy l iy w ih th ih n ih m s eh l f sil jh

./tmp-lstm/icefall_asr_timit_tdnn_lstm_ctc/test_waves/FELC0_SI756.WAV:
sil dh ih sil t ih r ih s sil s er r ih m ih sil m aa l ih ng sil k l ey sil r eh sil d␣
↪w ay sil d aa r sil b ah f sil <UNK> jh

./tmp-lstm/icefall_asr_timit_tdnn_lstm_ctc/test_waves/FMGD0_SI1564.WAV:
sil hh ae z sil b ih n iy w ah z sil b ae n ih sil b ay s sil n ey sil k eh l f eh n s␣
↪ih z eh n dh eh r w er sil g r ey z ih ng sil k ae dx l sil


2021-11-08 21:02:54,387 INFO [pretrained.py:269] Decoding Done
```

To decode with `whole-lattice-rescoring` methond, you can use

```
./tdnn_lstm_ctc/pretrained.py \
  --method whole-lattice-rescoring \
  --checkpoint ./tmp-lstm/icefall_asr_timit_tdnn_lstm_ctc/exp/pretrained_average_16_25.
↪pt \
  --words-file ./tmp-lstm/icefall_asr_timit_tdnn_lstm_ctc/data/lang_phone/words.txt \
  --HLG ./tmp-lstm/icefall_asr_timit_tdnn_lstm_ctc/data/lang_phone/HLG.pt \
  --G ./tmp-lstm/icefall_asr_timit_tdnn_lstm_ctc/data/lm/G_4_gram.pt \
  --ngram-lm-scale 0.08 \
  ./tmp-lstm/icefall_asr_timit_tdnn_lstm_ctc/test_waves/FDHC0_SI1559.WAV
  ./tmp-lstm/icefall_asr_timit_tdnn_lstm_ctc/test_waves/FELC0_SI756.WAV
  ./tmp-lstm/icefall_asr_timit_tdnn_lstm_ctc/test_waves/FMGD0_SI1564.WAV
```

The decoding output is:

```
2021-11-08 20:05:22,739 INFO [pretrained.py:169] device: cuda:0
2021-11-08 20:05:22,739 INFO [pretrained.py:171] Creating model
2021-11-08 20:05:26,959 INFO [pretrained.py:183] Loading HLG from ./tmp-lstm/icefall_asr_
```

(continues on next page)

(continued from previous page)

```
↪timit_tdnn_lstm_ctc/data/lang_phone/HLG.pt
2021-11-08 20:05:26,971 INFO [pretrained.py:191] Loading G from ./tmp-lstm/icefall_asr_
↪timit_tdnn_lstm_ctc/data/lm/G_4_gram.pt
2021-11-08 20:05:26,977 INFO [pretrained.py:200] Constructing Fbank computer
2021-11-08 20:05:26,978 INFO [pretrained.py:210] Reading sound files: ['./tmp-lstm/
↪icefall_asr_timit_tdnn_lstm_ctc/test_waves/FDHC0_SI1559.WAV', './tmp-lstm/icefall_asr_
↪timit_tdnn_lstm_ctc/test_waves/FELC0_SI756.WAV', './tmp-lstm/icefall_asr_timit_tdnn_
↪lstm_ctc/test_waves/FMGD0_SI1564.WAV']
2021-11-08 20:05:26,981 INFO [pretrained.py:216] Decoding started
2021-11-08 20:05:27,519 INFO [pretrained.py:251] Use HLG decoding + LM rescoring
2021-11-08 20:05:27,878 INFO [pretrained.py:267]
./tmp-lstm/icefall_asr_timit_tdnn_lstm_ctc/test_waves/FDHC0_SI1559.WAV:
sil dh ih sh uw l iy v iy z ih sil p r aa sil k s ah m ey dx ih sil w uh dx iy w ih s f␣
↪iy l ih ng w ih th ih n ih m s eh l f sil jh

./tmp-lstm/icefall_asr_timit_tdnn_lstm_ctc/test_waves/FELC0_SI756.WAV:
sil dh ih sil t ih r iy ih s sil s er r eh m ih sil n ah l ih ng sil k l ey sil r eh sil␣
↪d w ay sil d aa r sil b ow f sil jh

./tmp-lstm/icefall_asr_timit_tdnn_lstm_ctc/test_waves/FMGD0_SI1564.WAV:
sil hh ah z sil b ih n iy w ah z sil b ae n ih sil b ay s sil n ey sil k ih l f eh n s␣
↪ih z eh n dh eh r w er sil g r ey z ih n sil k ae dx l sil
```

2021-11-08 20:05:27,878 INFO [pretrained.py:269] Decoding Done

Colab notebook

We provide a colab notebook for decoding with pre-trained model.

Congratulations! You have finished the TDNN-LSTM-CTC recipe on timit in `icefall`.

6.1.4 YesNo

TDNN-CTC

This page shows you how to run the yesno recipe. It contains:

- (1) Prepare data for training
- (2) Train a TDNN model
  - (a) View text format logs and visualize TensorBoard logs
  - (b) Select device type, i.e., CPU and GPU, for training
  - (c) Change training options
  - (d) Resume training from a checkpoint
- (3) Decode with a trained model
  - (a) Select a checkpoint for decoding
  - (b) Model averaging

- (4) Colab notebook
  - (a) It shows you step by step how to setup the environment, how to do training, and how to do decoding
  - (b) How to use a pre-trained model
- (5) Inference with a pre-trained model
  - (a) Download a pre-trained model, provided by us
  - (b) Decode a single sound file with a pre-trained model
  - (c) Decode multiple sound files at the same time

It does **NOT** show you:

- (1) How to train with multiple GPUs

  The `yesno` dataset is so small that CPU is more than enough for training as well as for decoding.

- (2) How to use LM rescoring for decoding

  The dataset does not have an LM for rescoring.

Hint: We assume you have read the page *Installation* and have setup the environment for `icefall`.

Hint: You **don't** need a **GPU** to run this recipe. It can be run on a **CPU**. The training part takes less than 30 **seconds** on a CPU and you will get the following WER at the end:

```
[test_set] %WER 0.42% [1 / 240, 0 ins, 1 del, 0 sub ]
```

Data preparation

```
$ cd egs/yesno/ASR
$ ./prepare.sh
```

The script `./prepare.sh` handles the data preparation for you, **automagically**. All you need to do is to run it.

The data preparation contains several stages, you can use the following two options:

- `--stage`
- `--stop-stage`

to control which stage(s) should be run. By default, all stages are executed.

For example,

```
$ cd egs/yesno/ASR
$ ./prepare.sh --stage 0 --stop-stage 0
```

means to run only stage 0.

To run stage 2 to stage 5, use:

```
$ ./prepare.sh --stage 2 --stop-stage 5
```

Training

We provide only a TDNN model, contained in the tdnn folder, for `yesno`.

The command to run the training part is:

```
$ cd egs/yesno/ASR
$ export CUDA_VISIBLE_DEVICES=""
$ ./tdnn/train.py
```

By default, it will run 15 epochs. Training logs and checkpoints are saved in `tdnn/exp`.

In `tdnn/exp`, you will find the following files:

- `epoch-0.pt`, `epoch-1.pt`, ...

  These are checkpoint files, containing model `state_dict` and optimizer `state_dict`. To resume training from some checkpoint, say `epoch-10.pt`, you can use:

  ```
  $ ./tdnn/train.py --start-epoch 11
  ```

- `tensorboard/`

  This folder contains TensorBoard logs. Training loss, validation loss, learning rate, etc, are recorded in these logs. You can visualize them by:

  ```
  $ cd tdnn/exp/tensorboard
  $ tensorboard dev upload --logdir . --description "TDNN training for yesno␣
  ↪with icefall"
  ```

  It will print something like below:

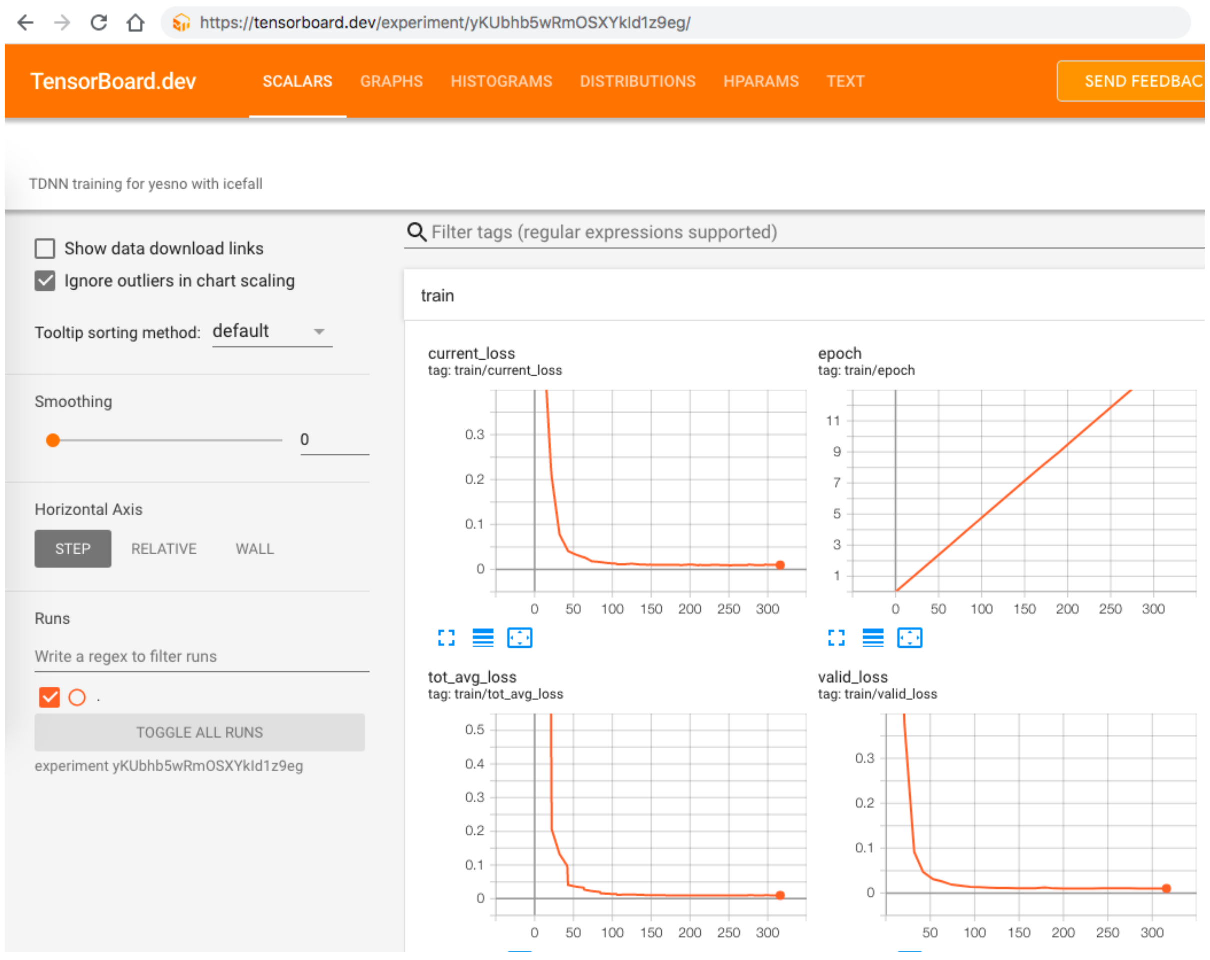
  ```
  TensorFlow installation not found - running with reduced feature set.
  Upload started and will continue reading any new data as it's added to the␣
  ↪logdir.

  To stop uploading, press Ctrl-C.

  New experiment created. View your TensorBoard at: https://tensorboard.dev/
  ↪experiment/yKUbhb5wRmOSXYkId1z9eg/

  [2021-08-23T23:49:41] Started scanning logdir.
  [2021-08-23T23:49:42] Total uploaded: 135 scalars, 0 tensors, 0 binary␣
  ↪objects
  Listening for new data in logdir...
  ```

  Note there is a URL in the above output, click it and you will see the following screenshot:

- `log/log-train-xxxx`

  It is the detailed training log in text format, same as the one you saw printed to the console during training.

Note: By default, `./tdnn/train.py` uses GPU 0 for training if GPUs are available. If you have two GPUs, say, GPU 0 and GPU 1, and you want to use GPU 1 for training, you can run:


https://tensorboard.dev/experiment/yKUbhb5wRmOSXYkld1z9eg/

TDNN training for yesno with icefall

Show data download links
Ignore outliers in chart scaling ✓
Tooltip sorting method: default
Smoothing: 0
Horizontal Axis: STEP [✓ STEP] RELATIVE WALL
Runs
Write a regex to filter runs
TOGGLE ALL RUNS
experiment yKUbhb5wRmOSXYkld1z9eg

Filter tags (regular expressions supported)

train

current_loss
tag: train/current_loss

epoch
tag: train/epoch

tot_avg_loss
tag: train/tot_avg_loss

valid_loss
tag: train/valid_loss

Fig. 6.8: TensorBoard screenshot.

```
$ export CUDA_VISIBLE_DEVICES="1"
$ ./tdnn/train.py
```

Since the `yesno` dataset is very small, containing only 30 sound files for training, and the model in use is also very small, we use:

```
$ export CUDA_VISIBLE_DEVICES=""
```

so that `./tdnn/train.py` uses CPU during training.

If you don't have GPUs, then you don't need to run `export CUDA_VISIBLE_DEVICES=""`.

To see available training options, you can use:

```
$ ./tdnn/train.py --help
```

Other training options, e.g., learning rate, results dir, etc., are pre-configured in the function `get_params()` in tdnn/train.py. Normally, you don't need to change them. You can change them by modifying the code, if you want.

Decoding

The decoding part uses checkpoints saved by the training part, so you have to run the training part first.

The command for decoding is:

```
$ export CUDA_VISIBLE_DEVICES=""
$ ./tdnn/decode.py
```

You will see the WER in the output log.

Decoded results are saved in `tdnn/exp`.

```
$ ./tdnn/decode.py --help
```

shows you the available decoding options.

Some commonly used options are:

- `--epoch`

  You can select which checkpoint to be used for decoding. For instance, `./tdnn/decode.py --epoch 10` means to use `./tdnn/exp/epoch-10.pt` for decoding.

- `--avg`

  It's related to model averaging. It specifies number of checkpoints to be averaged. The averaged model is used for decoding. For example, the following command:

  ```
  $ ./tdnn/decode.py --epoch 10 --avg 3
  ```

  uses the average of `epoch-8.pt`, `epoch-9.pt` and `epoch-10.pt` for decoding.

- `--export`

  If it is `True`, i.e., `./tdnn/decode.py --export 1`, the code will save the averaged model to `tdnn/exp/pretrained.pt`. See *Pre-trained Model* for how to use it.

Pre-trained Model

We have uploaded the pre-trained model to https://huggingface.co/csukuangfj/icefall_asr_yesno_tdnn.

The following shows you how to use the pre-trained model.

Download the pre-trained model

```
$ cd egs/yesno/ASR
$ mkdir tmp
$ cd tmp
$ git lfs install
$ git clone https://huggingface.co/csukuangfj/icefall_asr_yesno_tdnn
```

Caution: You have to use `git lfs` to download the pre-trained model.

After downloading, you will have the following files:

```
$ cd egs/yesno/ASR
$ tree tmp
```

```
tmp/
`-- icefall_asr_yesno_tdnn
    |-- README.md
    |-- lang_phone
    |   |-- HLG.pt
    |   |-- L.pt
    |   |-- L_disambig.pt
    |   |-- Linv.pt
    |   |-- lexicon.txt
    |   |-- lexicon_disambig.txt
    |   |-- tokens.txt
    |   `-- words.txt
    |-- lm
    |   |-- G.arpa
    |   `-- G.fst.txt
    |-- pretrained.pt
    `-- test_waves
        |-- 0_0_0_1_0_0_0_1.wav
        |-- 0_0_1_0_0_0_1_0.wav
        |-- 0_0_1_0_0_1_1_1.wav
        |-- 0_0_1_0_1_0_0_1.wav
        |-- 0_0_1_1_0_0_0_1.wav
        |-- 0_0_1_1_0_1_1_0.wav
        |-- 0_0_1_1_1_0_0_0.wav
        |-- 0_0_1_1_1_1_0_0.wav
        |-- 0_1_0_0_0_1_0_0.wav
        |-- 0_1_0_0_1_0_1_0.wav
        |-- 0_1_0_1_0_0_0_0.wav
        |-- 0_1_0_1_1_1_0_0.wav
```

(continues on next page)

(continued from previous page)

```
        |-- 0_1_1_0_0_1_1_1.wav
        |-- 0_1_1_1_0_0_1_0.wav
        |-- 0_1_1_1_1_0_1_0.wav
        |-- 1_0_0_0_0_0_0_0.wav
        |-- 1_0_0_0_0_0_1_1.wav
        |-- 1_0_0_1_0_1_1_1.wav
        |-- 1_0_1_1_0_1_1_1.wav
        |-- 1_0_1_1_1_1_0_1.wav
        |-- 1_1_0_0_0_1_1_1.wav
        |-- 1_1_0_0_1_0_1_1.wav
        |-- 1_1_0_1_0_1_0_0.wav
        |-- 1_1_0_1_1_0_0_1.wav
        |-- 1_1_0_1_1_1_1_0.wav
        |-- 1_1_1_0_0_1_0_1.wav
        |-- 1_1_1_0_1_0_1_0.wav
        |-- 1_1_1_1_0_0_1_0.wav
        |-- 1_1_1_1_1_0_0_0.wav
        `-- 1_1_1_1_1_1_1_1.wav

4 directories, 42 files
```

```
$ soxi tmp/icefall_asr_yesno_tdnn/test_waves/0_0_1_0_1_0_0_1.wav

Input File     : 'tmp/icefall_asr_yesno_tdnn/test_waves/0_0_1_0_1_0_0_1.wav'
Channels       : 1
Sample Rate    : 8000
Precision      : 16-bit
Duration       : 00:00:06.76 = 54080 samples ~ 507 CDDA sectors
File Size      : 108k
Bit Rate       : 128k
Sample Encoding: 16-bit Signed Integer PCM
```

- `0_0_1_0_1_0_0_1.wav`

  0 means No; 1 means Yes. No and Yes are not in English, but in Hebrew. So this file contains `NO NO YES NO YES NO NO YES`.

Download kaldifeat

kaldifeat is used for extracting features from a single or multiple sound files. Please refer to https://github.com/csukuangfj/kaldifeat to install `kaldifeat` first.

Inference with a pre-trained model

```
$ cd egs/yesno/ASR
$ ./tdnn/pretrained.py --help
```

shows the usage information of `./tdnn/pretrained.py`.

To decode a single file, we can use:

```
./tdnn/pretrained.py \
  --checkpoint ./tmp/icefall_asr_yesno_tdnn/pretrained.pt \
  --words-file ./tmp/icefall_asr_yesno_tdnn/lang_phone/words.txt \
  --HLG ./tmp/icefall_asr_yesno_tdnn/lang_phone/HLG.pt \
  ./tmp/icefall_asr_yesno_tdnn/test_waves/0_0_1_0_1_0_0_1.wav
```

The output is:

```
2021-08-24 12:22:51,621 INFO [pretrained.py:119] {'feature_dim': 23, 'num_classes': 4,
↪'sample_rate': 8000, 'search_beam': 20, 'output_beam': 8, 'min_active_states': 30,
↪'max_active_states': 10000, 'use_double_scores': True, 'checkpoint': './tmp/icefall_
↪asr_yesno_tdnn/pretrained.pt', 'words_file': './tmp/icefall_asr_yesno_tdnn/lang_phone/
↪words.txt', 'HLG': './tmp/icefall_asr_yesno_tdnn/lang_phone/HLG.pt', 'sound_files': ['.
↪/tmp/icefall_asr_yesno_tdnn/test_waves/0_0_1_0_1_0_0_1.wav']}
2021-08-24 12:22:51,645 INFO [pretrained.py:125] device: cpu
2021-08-24 12:22:51,645 INFO [pretrained.py:127] Creating model
2021-08-24 12:22:51,650 INFO [pretrained.py:139] Loading HLG from ./tmp/icefall_asr_
↪yesno_tdnn/lang_phone/HLG.pt
2021-08-24 12:22:51,651 INFO [pretrained.py:143] Constructing Fbank computer
2021-08-24 12:22:51,652 INFO [pretrained.py:153] Reading sound files: ['./tmp/icefall_
↪asr_yesno_tdnn/test_waves/0_0_1_0_1_0_0_1.wav']
2021-08-24 12:22:51,684 INFO [pretrained.py:159] Decoding started
2021-08-24 12:22:51,708 INFO [pretrained.py:198]
./tmp/icefall_asr_yesno_tdnn/test_waves/0_0_1_0_1_0_0_1.wav:
NO NO YES NO YES NO NO YES


2021-08-24 12:22:51,708 INFO [pretrained.py:200] Decoding Done
```

You can see that for the sound file `0_0_1_0_1_0_0_1.wav`, the decoding result is `NO NO YES NO YES NO NO YES`.

To decode **multiple** files at the same time, you can use

```
./tdnn/pretrained.py \
  --checkpoint ./tmp/icefall_asr_yesno_tdnn/pretrained.pt \
  --words-file ./tmp/icefall_asr_yesno_tdnn/lang_phone/words.txt \
  --HLG ./tmp/icefall_asr_yesno_tdnn/lang_phone/HLG.pt \
  ./tmp/icefall_asr_yesno_tdnn/test_waves/0_0_1_0_1_0_0_1.wav \
  ./tmp/icefall_asr_yesno_tdnn/test_waves/1_0_1_1_0_1_1_1.wav
```

The decoding output is:

```
2021-08-24 12:25:20,159 INFO [pretrained.py:119] {'feature_dim': 23, 'num_classes': 4,
↪'sample_rate': 8000, 'search_beam': 20, 'output_beam': 8, 'min_active_states': 30,
↪'max_active_states': 10000, 'use_double_scores': True, 'checkpoint': './tmp/icefall_
```

(continues on next page)

(continued from previous page)

```
↪asr_yesno_tdnn/pretrained.pt', 'words_file': './tmp/icefall_asr_yesno_tdnn/lang_phone/
↪words.txt', 'HLG': './tmp/icefall_asr_yesno_tdnn/lang_phone/HLG.pt', 'sound_files': ['.
↪/tmp/icefall_asr_yesno_tdnn/test_waves/0_0_1_0_1_0_0_1.wav', './tmp/icefall_asr_yesno_
↪tdnn/test_waves/1_0_1_1_0_1_1_1.wav']}
2021-08-24 12:25:20,181 INFO [pretrained.py:125] device: cpu
2021-08-24 12:25:20,181 INFO [pretrained.py:127] Creating model
2021-08-24 12:25:20,185 INFO [pretrained.py:139] Loading HLG from ./tmp/icefall_asr_
↪yesno_tdnn/lang_phone/HLG.pt
2021-08-24 12:25:20,186 INFO [pretrained.py:143] Constructing Fbank computer
2021-08-24 12:25:20,187 INFO [pretrained.py:153] Reading sound files: ['./tmp/icefall_
↪asr_yesno_tdnn/test_waves/0_0_1_0_1_0_0_1.wav',
'./tmp/icefall_asr_yesno_tdnn/test_waves/1_0_1_1_0_1_1_1.wav']
2021-08-24 12:25:20,213 INFO [pretrained.py:159] Decoding started
2021-08-24 12:25:20,287 INFO [pretrained.py:198]
./tmp/icefall_asr_yesno_tdnn/test_waves/0_0_1_0_1_0_0_1.wav:
NO NO YES NO YES NO NO YES

./tmp/icefall_asr_yesno_tdnn/test_waves/1_0_1_1_0_1_1_1.wav:
YES NO YES YES NO YES YES YES

2021-08-24 12:25:20,287 INFO [pretrained.py:200] Decoding Done
```

You can see again that it decodes correctly.

Colab notebook

We do provide a colab notebook for this recipe.

Congratulations! You have finished the simplest speech recognition recipe in `icefall`.

6.2 Streaming ASR

6.2.1 Introduction

This page shows you how we implement streaming **X-former transducer** models for ASR.

Hint: X-former transducer here means the encoder of the transducer model uses Multi-Head Attention, like Conformer, EmFormer etc.

Currently we have implemented two types of streaming models, one uses Conformer as encoder, the other uses Emformer as encoder.

Streaming Conformer

The main idea of training a streaming model is to make the model see limited contexts in training time, we can achieve this by applying a mask to the output of self-attention. In icefall, we implement the streaming conformer the way just like what WeNet did.

Note: The conformer-transducer recipes in LibriSpeech datasets, like, pruned_transducer_stateless, pruned_transducer_stateless2, pruned_transducer_stateless3, pruned_transducer_stateless4, pruned_transducer_stateless5 all support streaming.

Note: Training a streaming conformer model in `icefall` is almost the same as training a non-streaming model, all you need to do is passing several extra arguments. See *Pruned transducer statelessX* for more details.

Hint: If you want to modify a non-streaming conformer recipe to support both streaming and non-streaming, please refer to this pull request. After adding the code needed by streaming training, you have to re-train it with the extra arguments metioned in the docs above to get a streaming model.

Streaming Emformer

The Emformer model proposed here uses more complicated techniques. It has a memory bank component to memorize history information, what' more, it also introduces right context in training time by hard-copying part of the input features.

We have three variants of Emformer models in `icefall`.

- `pruned_stateless_emformer_rnnt2` using Emformer from torchaudio, see LibriSpeech recipe.
- `conv_emformer_transducer_stateless` using ConvEmformer implemented by ourself. Different from the Emformer in torchaudio, ConvEmformer has a convolution in each layer and uses the mechanisms in our reworked conformer model. See LibriSpeech recipe.
- `conv_emformer_transducer_stateless2` using ConvEmformer implemented by ourself. The only difference from the above one is that it uses a simplified memory bank. See LibriSpeech recipe.

6.2.2 LibriSpeech

Pruned transducer statelessX

This tutorial shows you how to run a **streaming** conformer transducer model with the LibriSpeech dataset.

Note: The tutorial is suitable for pruned_transducer_stateless, pruned_transducer_stateless2, pruned_transducer_stateless4, pruned_transducer_stateless5, We will take pruned_transducer_stateless4 as an example in this tutorial.

Hint: We assume you have read the page *Installation* and have setup the environment for `icefall`.

Hint: We recommend you to use a GPU or several GPUs to run this recipe.

Hint: Please scroll down to the bottom of this page to find download links for pretrained models if you don't want to train a model from scratch.

We use pruned RNN-T to compute the loss.

Note: You can find the paper about pruned RNN-T at the following address:

https://arxiv.org/abs/2206.13236

The transducer model consists of 3 parts:

- Encoder, a.k.a, the transcription network. We use a Conformer model (the reworked version by Daniel Povey)
- Decoder, a.k.a, the prediction network. We use a stateless model consisting of `nn.Embedding` and `nn.Conv1d`
- Joiner, a.k.a, the joint network.

Caution: Contrary to the conventional RNN-T models, we use a stateless decoder. That is, it has no recurrent connections.

Data preparation

Hint: The data preparation is the same as other recipes on LibriSpeech dataset, if you have finished this step, you can skip to `Training` directly.

```
$ cd egs/librispeech/ASR
$ ./prepare.sh
```

The script `./prepare.sh` handles the data preparation for you, **automagically**. All you need to do is to run it.

The data preparation contains several stages, you can use the following two options:

- `--stage`
- `--stop-stage`

to control which stage(s) should be run. By default, all stages are executed.

For example,

```
$ cd egs/librispeech/ASR
$ ./prepare.sh --stage 0 --stop-stage 0
```

means to run only stage 0.

To run stage 2 to stage 5, use:

```
$ ./prepare.sh --stage 2 --stop-stage 5
```

Hint: If you have pre-downloaded the LibriSpeech dataset and the musan dataset, say, they are saved in `/tmp/LibriSpeech` and `/tmp/musan`, you can modify the `dl_dir` variable in `./prepare.sh` to point to `/tmp` so that `./prepare.sh` won't re-download them.

Note: All generated files by `./prepare.sh`, e.g., features, lexicon, etc, are saved in `./data` directory.

We provide the following YouTube video showing how to run `./prepare.sh`.

Note: To get the latest news of next-gen Kaldi, please subscribe the following YouTube channel by Nadira Povey:

https://www.youtube.com/channel/UC_VaumpkmINz1pNkFXAN9mw

https://youtu.be/ofEIoJL-mGM

Training

Note: We put the streaming and non-streaming model in one recipe, to train a streaming model you only need to add **4** extra options comparing with training a non-streaming model. These options are `--dynamic-chunk-training`, `--num-left-chunks`, `--causal-convolution`, `--short-chunk-size`. You can see the configurable options below for their meanings or read https://arxiv.org/pdf/2012.05481.pdf for more details.

Configurable options

```
$ cd egs/librispeech/ASR
$ ./pruned_transducer_stateless4/train.py --help
```

shows you the training options that can be passed from the commandline. The following options are used quite often:

- `--exp-dir`

  The directory to save checkpoints, training logs and tensorboard.

- `--full-libri`

  If it's True, the training part uses all the training data, i.e., 960 hours. Otherwise, the training part uses only the subset `train-clean-100`, which has 100 hours of training data.

  Caution: The training set is perturbed by speed with two factors: 0.9 and 1.1. If `--full-libri` is True, each epoch actually processes `3x960 == 2880` hours of data.

- `--num-epochs`

  It is the number of epochs to train. For instance, `./pruned_transducer_stateless4/train.py --num-epochs 30` trains for 30 epochs and generates `epoch-1.pt`, `epoch-2.pt`, ..., `epoch-30.pt` in the folder `./pruned_transducer_stateless4/exp`.

- `--start-epoch`

  It's used to resume training. `./pruned_transducer_stateless4/train.py --start-epoch 10` loads the checkpoint `./pruned_transducer_stateless4/exp/epoch-9.pt` and starts training from epoch 10, based on the state from epoch 9.

- `--world-size`

  It is used for multi-GPU single-machine DDP training.

  - (a) If it is 1, then no DDP training is used.
  - (b) If it is 2, then GPU 0 and GPU 1 are used for DDP training.

  The following shows some use cases with it.

  Use case 1: You have 4 GPUs, but you only want to use GPU 0 and GPU 2 for training. You can do the following:

  ```
  $ cd egs/librispeech/ASR
  $ export CUDA_VISIBLE_DEVICES="0,2"
  $ ./pruned_transducer_stateless4/train.py --world-size 2
  ```

  Use case 2: You have 4 GPUs and you want to use all of them for training. You can do the following:

  ```
  $ cd egs/librispeech/ASR
  $ ./pruned_transducer_stateless4/train.py --world-size 4
  ```

  Use case 3: You have 4 GPUs but you only want to use GPU 3 for training. You can do the following:

  ```
  $ cd egs/librispeech/ASR
  $ export CUDA_VISIBLE_DEVICES="3"
  $ ./pruned_transducer_stateless4/train.py --world-size 1
  ```

Caution: Only multi-GPU single-machine DDP training is implemented at present. Multi-GPU multi-machine DDP training will be added later.

- `--max-duration`

  It specifies the number of seconds over all utterances in a batch, before **padding**. If you encounter CUDA OOM, please reduce it.

Hint: Due to padding, the number of seconds of all utterances in a batch will usually be larger than `--max-duration`.

A larger value for `--max-duration` may cause OOM during training, while a smaller value may increase the training time. You have to tune it.

- `--use-fp16`

  If it is True, the model will train with half precision, from our experiment results, by using half precision you can train with two times larger `--max-duration` so as to get almost 2X speed up.

- `--dynamic-chunk-training`

  The flag that indicates whether to train a streaming model or not, it **MUST** be True if you want to train a streaming model.

- `--short-chunk-size`

  When training a streaming attention model with chunk masking, the chunk size would be either max sequence length of current batch or uniformly sampled from (1, short_chunk_size). The default value is 25, you don't have to change it most of the time.

- `--num-left-chunks`

  It indicates how many left context (in chunks) that can be seen when calculating attention. The default value is 4, you don't have to change it most of the time.

- `--causal-convolution`

  Whether to use causal convolution in conformer encoder layer, this requires to be True when training a streaming model.

Pre-configured options

There are some training options, e.g., number of encoder layers, encoder dimension, decoder dimension, number of warmup steps etc, that are not passed from the commandline. They are pre-configured by the function `get_params()` in pruned_transducer_stateless4/train.py

You don't need to change these pre-configured parameters. If you really need to change them, please modify `./pruned_transducer_stateless4/train.py` directly.

Note: The options for pruned_transducer_stateless5 are a little different from other recipes. It allows you to configure `--num-encoder-layers`, `--dim-feedforward`, `--nhead`, `--encoder-dim`, `--decoder-dim`, `--joiner-dim` from commandline, so that you can train models with different size with pruned_transducer_stateless5.

Training logs

Training logs and checkpoints are saved in `--exp-dir` (e.g. `pruned_transducer_stateless4/exp`. You will find the following files in that directory:

- `epoch-1.pt`, `epoch-2.pt`, ...

  These are checkpoint files saved at the end of each epoch, containing model `state_dict` and optimizer `state_dict`. To resume training from some checkpoint, say `epoch-10.pt`, you can use:

  ```
  $ ./pruned_transducer_stateless4/train.py --start-epoch 11
  ```

- `checkpoint-436000.pt`, `checkpoint-438000.pt`, ...

  These are checkpoint files saved every `--save-every-n` batches, containing model `state_dict` and optimizer `state_dict`. To resume training from some checkpoint, say `checkpoint-436000`, you can use:

  ```
  $ ./pruned_transducer_stateless4/train.py --start-batch 436000
  ```

- `tensorboard/`

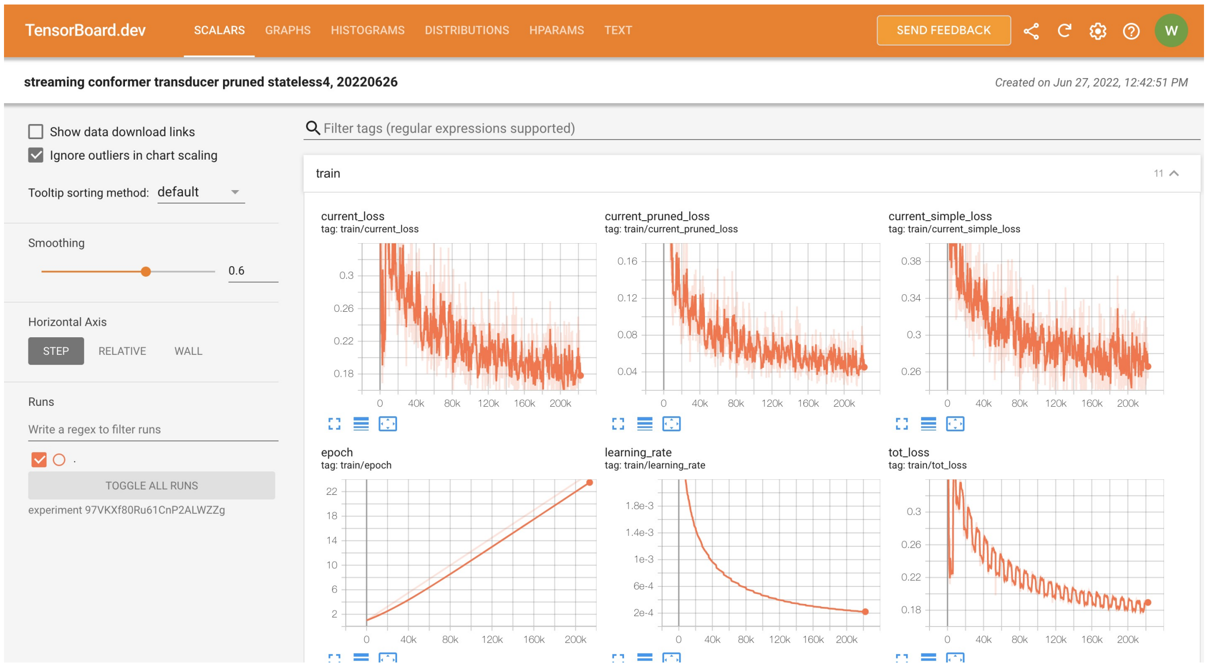
  This folder contains tensorBoard logs. Training loss, validation loss, learning rate, etc, are recorded in these logs. You can visualize them by:

```
$ cd pruned_transducer_stateless4/exp/tensorboard
$ tensorboard dev upload --logdir . --description "pruned␣
↪transducer training for LibriSpeech with icefall"
```

It will print something like below:

```
TensorFlow installation not found - running with reduced feature␣
↪set.
Upload started and will continue reading any new data as it's added␣
↪to the logdir.

To stop uploading, press Ctrl-C.

New experiment created. View your TensorBoard at: https://
↪tensorboard.dev/experiment/97VKXf80Ru61CnP2ALWZZg/

[2022-11-20T15:50:50] Started scanning logdir.
Uploading 4468 scalars...
[2022-11-20T15:53:02] Total uploaded: 210171 scalars, 0 tensors, 0␣
↪binary objects
Listening for new data in logdir...
```

Note there is a URL in the above output. Click it and you will see the following screenshot:



streaming conformer transducer pruned stateless4, 20220626

Created on Jun 27, 2022, 12:42:51 PM

☐ Show data download links
☑ Ignore outliers in chart scaling
Tooltip sorting method: default
Smoothing 0.6
Horizontal Axis: STEP RELATIVE WALL
Runs
Write a regex to filter runs
experiment 97VKXf80Ru61CnP2ALWZZg

train 11

current_loss tag: train/current_loss
current_pruned_loss tag: train/current_pruned_loss
current_simple_loss tag: train/current_simple_loss
epoch tag: train/epoch
learning_rate tag: train/learning_rate
tot_loss tag: train/tot_loss

Fig. 6.9: TensorBoard screenshot.

Hint: If you don't have access to google, you can use the following command to view the tensorboard log locally:

```
cd pruned_transducer_stateless4/exp/tensorboard
tensorboard --logdir . --port 6008
```

It will print the following message:

```
Serving TensorBoard on localhost; to expose to the network, use a␣
↪proxy or pass --bind_all
TensorBoard 2.8.0 at http://localhost:6008/ (Press CTRL+C to quit)
```

Now start your browser and go to http://localhost:6008 to view the tensorboard logs.

- `log/log-train-xxxx`

  It is the detailed training log in text format, same as the one you saw printed to the console during training.

Usage example

You can use the following command to start the training using 4 GPUs:

```
export CUDA_VISIBLE_DEVICES="0,1,2,3"
./pruned_transducer_stateless4/train.py \
   --world-size 4 \
   --dynamic-chunk-training 1 \
   --causal-convolution 1 \
   --num-epochs 30 \
   --start-epoch 1 \
   --exp-dir pruned_transducer_stateless4/exp \
   --full-libri 1 \
   --max-duration 300
```

Note: Comparing with training a non-streaming model, you only need to add two extra options, `--dynamic-chunk-training 1` and `--causal-convolution 1` .

Decoding

The decoding part uses checkpoints saved by the training part, so you have to run the training part first.

Hint: There are two kinds of checkpoints:

- (1) `epoch-1.pt`, `epoch-2.pt`, . . ., which are saved at the end of each epoch. You can pass `--epoch` to `pruned_transducer_stateless4/decode.py` to use them.
- (2) `checkpoints-436000.pt`, `epoch-438000.pt`, . . ., which are saved every `--save-every-n` batches. You can pass `--iter` to `pruned_transducer_stateless4/decode.py` to use them.

We suggest that you try both types of checkpoints and choose the one that produces the lowest WERs.

Tip: To decode a streaming model, you can use either `simulate streaming decoding` in `decode.py` or `real streaming decoding` in `streaming_decode.py`, the difference between `decode.py` and `streaming_decode.py` is that, `decode.py` processes the whole acoustic frames at one time with masking (i.e. same as training), but `streaming_decode.py` processes the acoustic frames chunk by chunk (so it can only see limited context).

Note: `simulate streaming decoding` in `decode.py` and `real streaming decoding` in `streaming_decode.py` should produce almost the same results given the same `--decode-chunk-size` and `--left-context`.

Simulate streaming decoding

```
$ cd egs/librispeech/ASR
$ ./pruned_transducer_stateless4/decode.py --help
```

shows the options for decoding. The following options are important for streaming models:

`--simulate-streaming`

If you want to decode a streaming model with `decode.py`, you **MUST** set `--simulate-streaming` to `True`. `simulate` here means the acoustic frames are not processed frame by frame (or chunk by chunk), instead, the whole sequence is processed at one time with masking (the same as training).

`--causal-convolution`

If True, the convolution module in encoder layers will be causal convolution. This is **MUST** be True when decoding with a streaming model.

`--decode-chunk-size`

For streaming models, we will calculate the chunk-wise attention, `--decode-chunk-size` indicates the chunk length (in frames after subsampling) for chunk-wise attention. For `simulate streaming decoding` the `decode-chunk-size` is used to generate the attention mask.

`--left-context`

`--left-context` indicates how many left context frames (after subsampling) can be seen for current chunk when calculating chunk-wise attention. Normally, `left-context` should equal to `decode-chunk-size * num-left-chunks`, where `num-left-chunks` is the option used to train this model. For `simulate streaming decoding` the `left-context` is used to generate the attention mask.

The following shows two examples (for the two types of checkpoints):

```
for m in greedy_search fast_beam_search modified_beam_search; do
  for epoch in 25 20; do
    for avg in 7 5 3 1; do
      ./pruned_transducer_stateless4/decode.py \
        --epoch $epoch \
        --avg $avg \
        --simulate-streaming 1 \
        --causal-convolution 1 \
        --decode-chunk-size 16 \
        --left-context 64 \
```

(continues on next page)

(continued from previous page)

```
        --exp-dir pruned_transducer_stateless4/exp \
        --max-duration 600 \
        --decoding-method $m
    done
  done
done
```

```
for m in greedy_search fast_beam_search modified_beam_search; do
  for iter in 474000; do
    for avg in 8 10 12 14 16 18; do
      ./pruned_transducer_stateless4/decode.py \
        --iter $iter \
        --avg $avg \
        --simulate-streaming 1 \
        --causal-convolution 1 \
        --decode-chunk-size 16 \
        --left-context 64 \
        --exp-dir pruned_transducer_stateless4/exp \
        --max-duration 600 \
        --decoding-method $m
    done
  done
done
```

Real streaming decoding

```
$ cd egs/librispeech/ASR
$ ./pruned_transducer_stateless4/streaming_decode.py --help
```

shows the options for decoding. The following options are important for streaming models:

`--decode-chunk-size`

For streaming models, we will calculate the chunk-wise attention, `--decode-chunk-size` indicates the chunk length (in frames after subsampling) for chunk-wise attention. For `real streaming decoding`, we will process `decode-chunk-size` acoustic frames at each time.

`--left-context`

`--left-context` indicates how many left context frames (after subsampling) can be seen for current chunk when calculating chunk-wise attention. Normally, `left-context` should equal to `decode-chunk-size * num-left-chunks`, where `num-left-chunks` is the option used to train this model.

`--num-decode-streams`

The number of decoding streams that can be run in parallel (very similar to the `bath size`). For `real streaming decoding`, the batches will be packed dynamically, for example, if the `num-decode-streams` equals to 10, then, sequence 1 to 10 will be decoded at first, after a while, suppose sequence 1 and 2 are done, so, sequence 3 to 12 will be processed parallelly in a batch.

Note: We also try adding `--right-context` in the real streaming decoding, but it seems not to benefit the performance for all the models, the reasons might be the training and decoding mismatch. You can try decoding with `--right-context` to see if it helps. The default value is 0.

The following shows two examples (for the two types of checkpoints):

```
for m in greedy_search fast_beam_search modified_beam_search; do
  for epoch in 25 20; do
    for avg in 7 5 3 1; do
      ./pruned_transducer_stateless4/decode.py \
        --epoch $epoch \
        --avg $avg \
        --decode-chunk-size 16 \
        --left-context 64 \
        --num-decode-streams 100 \
        --exp-dir pruned_transducer_stateless4/exp \
        --max-duration 600 \
        --decoding-method $m
    done
  done
done
```

```
for m in greedy_search fast_beam_search modified_beam_search; do
  for iter in 474000; do
    for avg in 8 10 12 14 16 18; do
      ./pruned_transducer_stateless4/decode.py \
        --iter $iter \
        --avg $avg \
        --decode-chunk-size 16 \
        --left-context 64 \
        --num-decode-streams 100 \
        --exp-dir pruned_transducer_stateless4/exp \
        --max-duration 600 \
        --decoding-method $m
    done
  done
done
```

Tip: Supporting decoding methods are as follows:

- `greedy_search` : It takes the symbol with largest posterior probability of each frame as the decoding result.
- `beam_search` : It implements Algorithm 1 in https://arxiv.org/pdf/1211.3711.pdf and espnet/nets/beam_search_transducer.py is used as a reference. Basicly, it keeps topk states for each frame, and expands the kept states with their own contexts to next frame.
- `modified_beam_search` : It implements the same algorithm as `beam_search` above, but it runs in batch mode with `--max-sym-per-frame=1` being hardcoded.
- `fast_beam_search` : It implements graph composition between the output `log_probs` and given `FSAs`. It is hard to describe the details in several lines of texts, you can read our paper in https://arxiv.org/pdf/2211.00484.pdf or our rnnt decode code in k2. `fast_beam_search` can decode with `FSAs` on GPU efficiently.
- `fast_beam_search_LG` : The same as `fast_beam_search` above, `fast_beam_search` uses an trivial graph that has only one state, while `fast_beam_search_LG` uses an LG graph (with N-gram LM).

- `fast_beam_search_nbest` : It produces the decoding results as follows:
  - (1) Use `fast_beam_search` to get a lattice
  - (2) Select `num_paths` paths from the lattice using `k2.random_paths()`
  - (3) Unique the selected paths
  - (4) Intersect the selected paths with the lattice and compute the shortest path from the intersection result
  - (5) The path with the largest score is used as the decoding output.
- `fast_beam_search_nbest_LG` : It implements same logic as `fast_beam_search_nbest`, the only difference is that it uses `fast_beam_search_LG` to generate the lattice.

Note: The supporting decoding methods in `streaming_decode.py` might be less than that in `decode.py`, if needed, you can implement them by yourself or file a issue in icefall .

Export Model

pruned_transducer_stateless4/export.py supports exporting checkpoints from `pruned_transducer_stateless4/exp` in the following ways.

Export `model.state_dict()`

Checkpoints saved by `pruned_transducer_stateless4/train.py` also include `optimizer.state_dict()`. It is useful for resuming training. But after training, we are interested only in `model.state_dict()`. You can use the following command to extract `model.state_dict()`.

```
# Assume that --epoch 25 --avg 3 produces the smallest WER
# (You can get such information after running ./pruned_transducer_stateless4/decode.py)

epoch=25
avg=3

./pruned_transducer_stateless4/export.py \
  --exp-dir ./pruned_transducer_stateless4/exp \
  --streaming-model 1 \
  --causal-convolution 1 \
  --bpe-model data/lang_bpe_500/bpe.model \
  --epoch $epoch \
  --avg  $avg
```

Caution: `--streaming-model` and `--causal-convolution` require to be True to export a streaming mdoel.

It will generate a file `./pruned_transducer_stateless4/exp/pretrained.pt`.

Hint: To use the generated `pretrained.pt` for `pruned_transducer_stateless4/decode.py`, you can run:

```
cd pruned_transducer_stateless4/exp
ln -s pretrained.pt epoch-999.pt
```

And then pass `--epoch 999 --avg 1 --use-averaged-model 0` to `./pruned_transducer_stateless4/decode.py`.

To use the exported model with `./pruned_transducer_stateless4/pretrained.py`, you can run:

```
./pruned_transducer_stateless4/pretrained.py \
  --checkpoint ./pruned_transducer_stateless4/exp/pretrained.pt \
  --simulate-streaming 1 \
  --causal-convolution 1 \
  --bpe-model ./data/lang_bpe_500/bpe.model \
  --method greedy_search \
  /path/to/foo.wav \
  /path/to/bar.wav
```

Export model using `torch.jit.script()`

```
./pruned_transducer_stateless4/export.py \
  --exp-dir ./pruned_transducer_stateless4/exp \
  --streaming-model 1 \
  --causal-convolution 1 \
  --bpe-model data/lang_bpe_500/bpe.model \
  --epoch 25 \
  --avg 3 \
  --jit 1
```

Caution: `--streaming-model` and `--causal-convolution` require to be True to export a streaming mdoel.

It will generate a file `cpu_jit.pt` in the given `exp_dir`. You can later load it by `torch.jit.load("cpu_jit.pt")`.

Note `cpu` in the name `cpu_jit.pt` means the parameters when loaded into Python are on CPU. You can use `to("cuda")` to move them to a CUDA device.

Note: You will need this `cpu_jit.pt` when deploying with Sherpa framework.

Download pretrained models

If you don't want to train from scratch, you can download the pretrained models by visiting the following links:

- pruned_transducer_stateless
- pruned_transducer_stateless2
- pruned_transducer_stateless4
- pruned_transducer_stateless5

See https://github.com/k2-fsa/icefall/blob/master/egs/librispeech/ASR/RESULTS.md for the details of the above pretrained models

Deploy with Sherpa

Please see https://k2-fsa.github.io/sherpa/python/streaming_asr/conformer/index.html# for how to deploy the models in `sherpa`.

LSTM Transducer

Hint: Please scroll down to the bottom of this page to find download links for pretrained models if you don't want to train a model from scratch.

This tutorial shows you how to train an LSTM transducer model with the LibriSpeech dataset.

We use pruned RNN-T to compute the loss.

Note: You can find the paper about pruned RNN-T at the following address:

https://arxiv.org/abs/2206.13236

The transducer model consists of 3 parts:

- Encoder, a.k.a, the transcription network. We use an LSTM model
- Decoder, a.k.a, the prediction network. We use a stateless model consisting of `nn.Embedding` and `nn.Conv1d`
- Joiner, a.k.a, the joint network.

Caution: Contrary to the conventional RNN-T models, we use a stateless decoder. That is, it has no recurrent connections.

Hint: Since the encoder model is an LSTM, not Transformer/Conformer, the resulting model is suitable for streaming/online ASR.

Which model to use

Currently, there are two folders about LSTM stateless transducer training:

- `(1)` https://github.com/k2-fsa/icefall/tree/master/egs/librispeech/ASR/lstm_transducer_stateless

  This recipe uses only LibriSpeech during training.

- `(2)` https://github.com/k2-fsa/icefall/tree/master/egs/librispeech/ASR/lstm_transducer_stateless2

  This recipe uses GigaSpeech + LibriSpeech during training.

`(1)` and `(2)` use the same model architecture. The only difference is that `(2)` supports multi-dataset. Since `(2)` uses more data, it has a lower WER than `(1)` but it needs more training time.

We use `lstm_transducer_stateless2` as an example below.

Note: You need to download the GigaSpeech dataset to run (2). If you have only LibriSpeech dataset available, feel free to use (1).

Data preparation

```
$ cd egs/librispeech/ASR
$ ./prepare.sh

# If you use (1), you can **skip** the following command
$ ./prepare_giga_speech.sh
```

The script ./prepare.sh handles the data preparation for you, **automagically**. All you need to do is to run it.

Note: We encourage you to read ./prepare.sh.

The data preparation contains several stages. You can use the following two options:

- --stage
- --stop-stage

to control which stage(s) should be run. By default, all stages are executed.

For example,

```
$ cd egs/librispeech/ASR
$ ./prepare.sh --stage 0 --stop-stage 0
```

means to run only stage 0.

To run stage 2 to stage 5, use:

```
$ ./prepare.sh --stage 2 --stop-stage 5
```

Hint: If you have pre-downloaded the LibriSpeech dataset and the musan dataset, say, they are saved in /tmp/LibriSpeech and /tmp/musan, you can modify the dl_dir variable in ./prepare.sh to point to /tmp so that ./prepare.sh won't re-download them.

Note: All generated files by ./prepare.sh, e.g., features, lexicon, etc, are saved in ./data directory.

We provide the following YouTube video showing how to run ./prepare.sh.

Note: To get the latest news of next-gen Kaldi, please subscribe the following YouTube channel by Nadira Povey:

https://www.youtube.com/channel/UC_VaumpkmINz1pNkFXAN9mw

https://youtu.be/ofEIoJL-mGM

Training

Configurable options

```
$ cd egs/librispeech/ASR
$ ./lstm_transducer_stateless2/train.py --help
```

shows you the training options that can be passed from the commandline. The following options are used quite often:

- `--full-libri`

  If it's True, the training part uses all the training data, i.e., 960 hours. Otherwise, the training part uses only the subset `train-clean-100`, which has 100 hours of training data.

  > **Caution:** The training set is perturbed by speed with two factors: 0.9 and 1.1. If `--full-libri` is True, each epoch actually processes `3x960 == 2880` hours of data.

- `--num-epochs`

  It is the number of epochs to train. For instance, `./lstm_transducer_stateless2/train.py --num-epochs 30` trains for 30 epochs and generates `epoch-1.pt`, `epoch-2.pt`, . . ., `epoch-30.pt` in the folder `./lstm_transducer_stateless2/exp`.

- `--start-epoch`

  It's used to resume training. `./lstm_transducer_stateless2/train.py --start-epoch 10` loads the checkpoint `./lstm_transducer_stateless2/exp/epoch-9.pt` and starts training from epoch 10, based on the state from epoch 9.

- `--world-size`

  It is used for multi-GPU single-machine DDP training.

  - (a) If it is 1, then no DDP training is used.
  - (b) If it is 2, then GPU 0 and GPU 1 are used for DDP training.

  The following shows some use cases with it.

  Use case 1: You have 4 GPUs, but you only want to use GPU 0 and GPU 2 for training. You can do the following:

  ```
  $ cd egs/librispeech/ASR
  $ export CUDA_VISIBLE_DEVICES="0,2"
  $ ./lstm_transducer_stateless2/train.py --world-size 2
  ```

  Use case 2: You have 4 GPUs and you want to use all of them for training. You can do the following:

  ```
  $ cd egs/librispeech/ASR
  $ ./lstm_transducer_stateless2/train.py --world-size 4
  ```

  Use case 3: You have 4 GPUs but you only want to use GPU 3 for training. You can do the following:

  ```
  $ cd egs/librispeech/ASR
  $ export CUDA_VISIBLE_DEVICES="3"
  $ ./lstm_transducer_stateless2/train.py --world-size 1
  ```

Caution: Only multi-GPU single-machine DDP training is implemented at present. Multi-GPU multi-machine DDP training will be added later.

- `--max-duration`

  It specifies the number of seconds over all utterances in a batch, before **padding**. If you encounter CUDA OOM, please reduce it.

  Hint: Due to padding, the number of seconds of all utterances in a batch will usually be larger than `--max-duration`.

  A larger value for `--max-duration` may cause OOM during training, while a smaller value may increase the training time. You have to tune it.

- `--giga-prob`

  The probability to select a batch from the `GigaSpeech` dataset. Note: It is available only for `(2)`.

Pre-configured options

There are some training options, e.g., weight decay, number of warmup steps, results dir, etc, that are not passed from the commandline. They are pre-configured by the function `get_params()` in lstm_transducer_stateless2/train.py

You don't need to change these pre-configured parameters. If you really need to change them, please modify `./lstm_transducer_stateless2/train.py` directly.

Training logs

Training logs and checkpoints are saved in `lstm_transducer_stateless2/exp`. You will find the following files in that directory:

- `epoch-1.pt`, `epoch-2.pt`, ...

  These are checkpoint files saved at the end of each epoch, containing model `state_dict` and optimizer `state_dict`. To resume training from some checkpoint, say `epoch-10.pt`, you can use:

  ```
  $ ./lstm_transducer_stateless2/train.py --start-epoch 11
  ```

- `checkpoint-436000.pt`, `checkpoint-438000.pt`, ...

  These are checkpoint files saved every `--save-every-n` batches, containing model `state_dict` and optimizer `state_dict`. To resume training from some checkpoint, say `checkpoint-436000`, you can use:

  ```
  $ ./lstm_transducer_stateless2/train.py --start-batch 436000
  ```

- `tensorboard/`

  This folder contains tensorBoard logs. Training loss, validation loss, learning rate, etc, are recorded in these logs. You can visualize them by:

  ```
  $ cd lstm_transducer_stateless2/exp/tensorboard
  $ tensorboard dev upload --logdir . --description "LSTM transducer␣
  ↪training for LibriSpeech with icefall"
  ```

It will print something like below:

```
TensorFlow installation not found - running with reduced feature␣
↪set.
Upload started and will continue reading any new data as it's added␣
↪to the logdir.

To stop uploading, press Ctrl-C.

New experiment created. View your TensorBoard at: https://
↪tensorboard.dev/experiment/cj2vtPiwQHKN9Q1tx6PTpg/

[2022-09-20T15:50:50] Started scanning logdir.
Uploading 4468 scalars...
[2022-09-20T15:53:02] Total uploaded: 210171 scalars, 0 tensors, 0␣
↪binary objects
Listening for new data in logdir...
```

Note there is a URL in the above output. Click it and you will see the following screenshot:

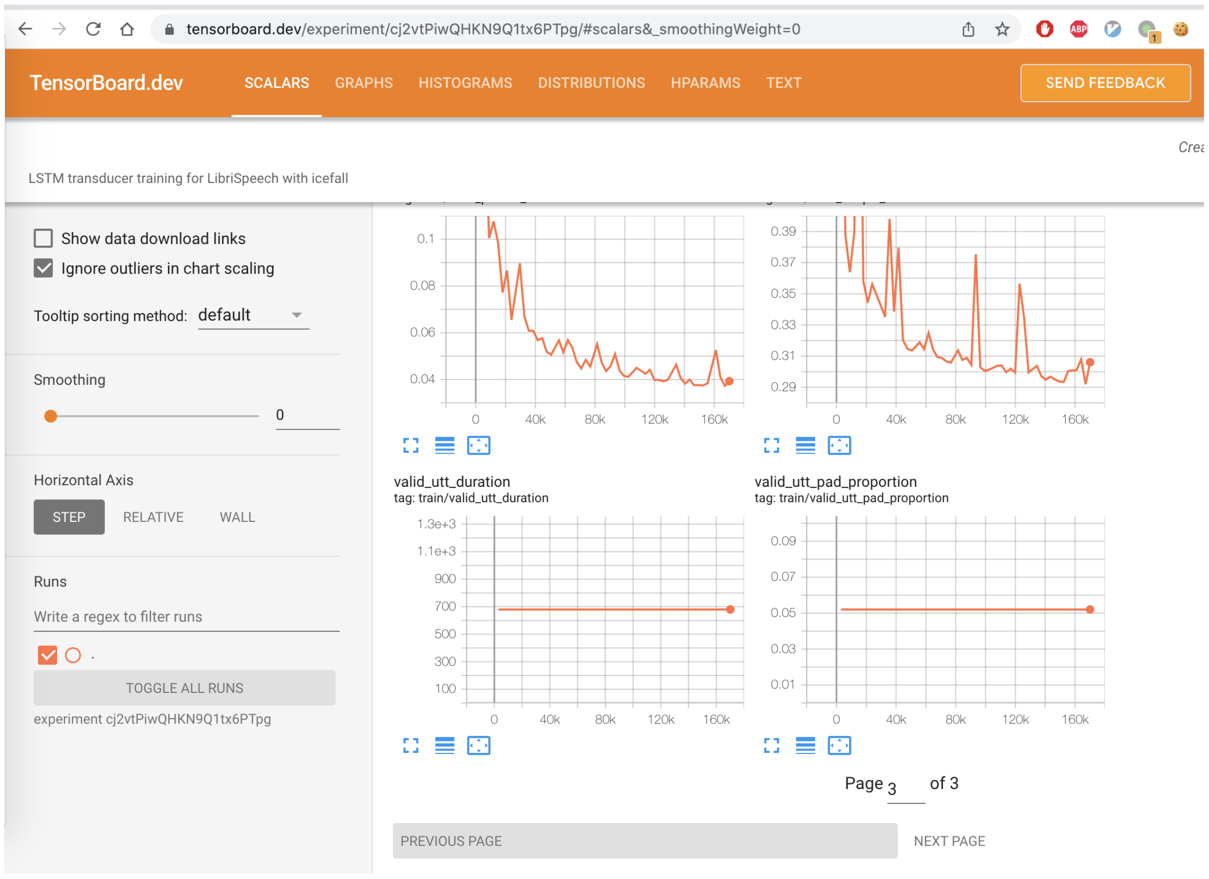

tensorboard.dev/experiment/cj2vtPiwQHKN9Q1tx6PTpg/#scalars&_smoothingWeight=0
TensorBoard.dev
SCALARS
GRAPHS
HISTOGRAMS
DISTRIBUTIONS
HPARAMS
TEXT
SEND FEEDBACK
LSTM transducer training for LibriSpeech with icefall
Show data download links
Ignore outliers in chart scaling
Tooltip sorting method: default
Smoothing
0
Horizontal Axis
STEP
RELATIVE
WALL
Runs
Write a regex to filter runs
TOGGLE ALL RUNS
experiment cj2vtPiwQHKN9Q1tx6PTpg
valid_utt_duration
tag: train/valid_utt_duration
valid_utt_pad_proportion
tag: train/valid_utt_pad_proportion
Page 3 of 3
PREVIOUS PAGE
NEXT PAGE

Fig. 6.10: TensorBoard screenshot.

Hint: If you don't have access to google, you can use the following command to view the tensorboard log locally:

```
cd lstm_transducer_stateless2/exp/tensorboard
tensorboard --logdir . --port 6008
```

It will print the following message:

```
Serving TensorBoard on localhost; to expose to the network, use a␣
↪proxy or pass --bind_all
TensorBoard 2.8.0 at http://localhost:6008/ (Press CTRL+C to quit)
```

Now start your browser and go to http://localhost:6008 to view the tensorboard logs.

- `log/log-train-xxxx`

  It is the detailed training log in text format, same as the one you saw printed to the console during training.

Usage example

You can use the following command to start the training using 8 GPUs:

```
export CUDA_VISIBLE_DEVICES="0,1,2,3,4,5,6,7"
./lstm_transducer_stateless2/train.py \
  --world-size 8 \
  --num-epochs 35 \
  --start-epoch 1 \
  --full-libri 1 \
  --exp-dir lstm_transducer_stateless2/exp \
  --max-duration 500 \
  --use-fp16 0 \
  --lr-epochs 10 \
  --num-workers 2 \
  --giga-prob 0.9
```

Decoding

The decoding part uses checkpoints saved by the training part, so you have to run the training part first.

Hint: There are two kinds of checkpoints:

- (1) `epoch-1.pt`, `epoch-2.pt`, . . . , which are saved at the end of each epoch. You can pass `--epoch` to `lstm_transducer_stateless2/decode.py` to use them.
- (2) `checkpoints-436000.pt`, `epoch-438000.pt`, . . . , which are saved every `--save-every-n` batches. You can pass `--iter` to `lstm_transducer_stateless2/decode.py` to use them.

We suggest that you try both types of checkpoints and choose the one that produces the lowest WERs.

```
$ cd egs/librispeech/ASR
$ ./lstm_transducer_stateless2/decode.py --help
```

shows the options for decoding.

The following shows two examples:

```
for m in greedy_search fast_beam_search modified_beam_search; do
  for epoch in 17; do
    for avg in 1 2; do
      ./lstm_transducer_stateless2/decode.py \
        --epoch $epoch \
        --avg $avg \
        --exp-dir lstm_transducer_stateless2/exp \
        --max-duration 600 \
        --num-encoder-layers 12 \
        --rnn-hidden-size 1024 \
        --decoding-method $m \
        --use-averaged-model True \
        --beam 4 \
        --max-contexts 4 \
        --max-states 8 \
        --beam-size 4
    done
  done
done
```

```
for m in greedy_search fast_beam_search modified_beam_search; do
  for iter in 474000; do
    for avg in 8 10 12 14 16 18; do
      ./lstm_transducer_stateless2/decode.py \
        --iter $iter \
        --avg $avg \
        --exp-dir lstm_transducer_stateless2/exp \
        --max-duration 600 \
        --num-encoder-layers 12 \
        --rnn-hidden-size 1024 \
        --decoding-method $m \
        --use-averaged-model True \
        --beam 4 \
        --max-contexts 4 \
        --max-states 8 \
        --beam-size 4
    done
  done
done
```

Export models

lstm_transducer_stateless2/export.py supports exporting checkpoints from `lstm_transducer_stateless2/exp` in the following ways.

Export `model.state_dict()`

Checkpoints saved by `lstm_transducer_stateless2/train.py` also include `optimizer.state_dict()`. It is useful for resuming training. But after training, we are interested only in `model.state_dict()`. You can use the following command to extract `model.state_dict()`.

```
# Assume that --iter 468000 --avg 16 produces the smallest WER
# (You can get such information after running ./lstm_transducer_stateless2/decode.py)

iter=468000
avg=16

./lstm_transducer_stateless2/export.py \
  --exp-dir ./lstm_transducer_stateless2/exp \
  --bpe-model data/lang_bpe_500/bpe.model \
  --iter $iter \
  --avg  $avg
```

It will generate a file `./lstm_transducer_stateless2/exp/pretrained.pt`.

Hint: To use the generated `pretrained.pt` for `lstm_transducer_stateless2/decode.py`, you can run:

```
cd lstm_transducer_stateless2/exp
ln -s pretrained epoch-9999.pt
```

And then pass `--epoch 9999 --avg 1 --use-averaged-model 0` to `./lstm_transducer_stateless2/decode.py`.

To use the exported model with `./lstm_transducer_stateless2/pretrained.py`, you can run:

```
./lstm_transducer_stateless2/pretrained.py \
  --checkpoint ./lstm_transducer_stateless2/exp/pretrained.pt \
  --bpe-model ./data/lang_bpe_500/bpe.model \
  --method greedy_search \
  /path/to/foo.wav \
  /path/to/bar.wav
```

Export model using `torch.jit.trace()`

```
iter=468000
avg=16

./lstm_transducer_stateless2/export.py \
  --exp-dir ./lstm_transducer_stateless2/exp \
  --bpe-model data/lang_bpe_500/bpe.model \
  --iter $iter \
  --avg  $avg \
  --jit-trace 1
```

It will generate 3 files:

- `./lstm_transducer_stateless2/exp/encoder_jit_trace.pt`
- `./lstm_transducer_stateless2/exp/decoder_jit_trace.pt`
- `./lstm_transducer_stateless2/exp/joiner_jit_trace.pt`

To use the generated files with `./lstm_transducer_stateless2/jit_pretrained`:

```
./lstm_transducer_stateless2/jit_pretrained.py \
  --bpe-model ./data/lang_bpe_500/bpe.model \
  --encoder-model-filename ./lstm_transducer_stateless2/exp/encoder_jit_trace.pt \
  --decoder-model-filename ./lstm_transducer_stateless2/exp/decoder_jit_trace.pt \
  --joiner-model-filename ./lstm_transducer_stateless2/exp/joiner_jit_trace.pt \
  /path/to/foo.wav \
  /path/to/bar.wav
```

Hint: Please see https://k2-fsa.github.io/sherpa/python/streaming_asr/lstm/english/server.html for how to use the exported models in `sherpa`.

Download pretrained models

If you don't want to train from scratch, you can download the pretrained models by visiting the following links:

- https://huggingface.co/csukuangfj/icefall-asr-librispeech-lstm-transducer-stateless2-2022-09-03
- https://huggingface.co/Zengwei/icefall-asr-librispeech-lstm-transducer-stateless-2022-08-18

See https://github.com/k2-fsa/icefall/blob/master/egs/librispeech/ASR/RESULTS.md for the details of the above pretrained models

You can find more usages of the pretrained models in https://k2-fsa.github.io/sherpa/python/streaming_asr/lstm/index.html

Zipformer Transducer

This tutorial shows you how to run a **streaming** zipformer transducer model with the LibriSpeech dataset.

Note: The tutorial is suitable for pruned_transducer_stateless7_streaming,

Hint: We assume you have read the page *Installation* and have setup the environment for `icefall`.

Hint: We recommend you to use a GPU or several GPUs to run this recipe.

Hint: Please scroll down to the bottom of this page to find download links for pretrained models if you don't want to train a model from scratch.

We use pruned RNN-T to compute the loss.

Note: You can find the paper about pruned RNN-T at the following address:

https://arxiv.org/abs/2206.13236

The transducer model consists of 3 parts:

- Encoder, a.k.a, the transcription network. We use a Zipformer model (proposed by Daniel Povey)
- Decoder, a.k.a, the prediction network. We use a stateless model consisting of `nn.Embedding` and `nn.Conv1d`
- Joiner, a.k.a, the joint network.

Caution: Contrary to the conventional RNN-T models, we use a stateless decoder. That is, it has no recurrent connections.

Data preparation

Hint: The data preparation is the same as other recipes on LibriSpeech dataset, if you have finished this step, you can skip to `Training` directly.

```
$ cd egs/librispeech/ASR
$ ./prepare.sh
```

The script `./prepare.sh` handles the data preparation for you, **automagically**. All you need to do is to run it.

The data preparation contains several stages, you can use the following two options:

- `--stage`
- `--stop-stage`

to control which stage(s) should be run. By default, all stages are executed.

For example,

```
$ cd egs/librispeech/ASR
$ ./prepare.sh --stage 0 --stop-stage 0
```

means to run only stage 0.

To run stage 2 to stage 5, use:

```
$ ./prepare.sh --stage 2 --stop-stage 5
```

Hint: If you have pre-downloaded the LibriSpeech dataset and the musan dataset, say, they are saved in `/tmp/LibriSpeech` and `/tmp/musan`, you can modify the `dl_dir` variable in `./prepare.sh` to point to `/tmp` so that `./prepare.sh` won't re-download them.

Note: All generated files by `./prepare.sh`, e.g., features, lexicon, etc, are saved in `./data` directory.

We provide the following YouTube video showing how to run `./prepare.sh`.

Note: To get the latest news of next-gen Kaldi, please subscribe the following YouTube channel by Nadira Povey:

https://www.youtube.com/channel/UC_VaumpkmINz1pNkFXAN9mw

https://youtu.be/ofEIoJL-mGM

Training

Configurable options

```
$ cd egs/librispeech/ASR
$ ./pruned_transducer_stateless7_streaming/train.py --help
```

shows you the training options that can be passed from the commandline. The following options are used quite often:

- `--exp-dir`

  The directory to save checkpoints, training logs and tensorboard.

- `--full-libri`

  If it's True, the training part uses all the training data, i.e., 960 hours. Otherwise, the training part uses only the subset `train-clean-100`, which has 100 hours of training data.

  Caution: The training set is perturbed by speed with two factors: 0.9 and 1.1. If `--full-libri` is True, each epoch actually processes `3x960 == 2880` hours of data.

- `--num-epochs`

It is the number of epochs to train. For instance, `./pruned_transducer_stateless7_streaming/train.py --num-epochs 30` trains for 30 epochs and generates `epoch-1.pt`, `epoch-2.pt`, . . . , `epoch-30.pt` in the folder `./pruned_transducer_stateless7_streaming/exp`.

- `--start-epoch`

  It's used to resume training. `./pruned_transducer_stateless7_streaming/train.py --start-epoch 10` loads the checkpoint `./pruned_transducer_stateless7_streaming/exp/epoch-9.pt` and starts training from epoch 10, based on the state from epoch 9.

- `--world-size`

  It is used for multi-GPU single-machine DDP training.

  - (a) If it is 1, then no DDP training is used.
  - (b) If it is 2, then GPU 0 and GPU 1 are used for DDP training.

  The following shows some use cases with it.

  Use case 1: You have 4 GPUs, but you only want to use GPU 0 and GPU 2 for training. You can do the following:

```
$ cd egs/librispeech/ASR
$ export CUDA_VISIBLE_DEVICES="0,2"
$ ./pruned_transducer_stateless7_streaming/train.py --world-size 2
```

  Use case 2: You have 4 GPUs and you want to use all of them for training. You can do the following:

```
$ cd egs/librispeech/ASR
$ ./pruned_transducer_stateless7_streaming/train.py --world-size 4
```

  Use case 3: You have 4 GPUs but you only want to use GPU 3 for training. You can do the following:

```
$ cd egs/librispeech/ASR
$ export CUDA_VISIBLE_DEVICES="3"
$ ./pruned_transducer_stateless7_streaming/train.py --world-size 1
```

Caution: Only multi-GPU single-machine DDP training is implemented at present. Multi-GPU multi-machine DDP training will be added later.

- `--max-duration`

  It specifies the number of seconds over all utterances in a batch, before **padding**. If you encounter CUDA OOM, please reduce it.

Hint: Due to padding, the number of seconds of all utterances in a batch will usually be larger than `--max-duration`.

A larger value for `--max-duration` may cause OOM during training, while a smaller value may increase the training time. You have to tune it.

- `--use-fp16`

  If it is True, the model will train with half precision, from our experiment results, by using half precision you can train with two times larger `--max-duration` so as to get almost 2X speed up.

  We recommend using `--use-fp16 True`.

- `--short-chunk-size`

  When training a streaming attention model with chunk masking, the chunk size would be either max sequence length of current batch or uniformly sampled from (1, short_chunk_size). The default value is 50, you don't have to change it most of the time.

- `--num-left-chunks`

  It indicates how many left context (in chunks) that can be seen when calculating attention. The default value is 4, you don't have to change it most of the time.

- `--decode-chunk-len`

  The chunk size for decoding (in frames before subsampling). It is used for validation. The default value is 32 (i.e., 320ms).

Pre-configured options

There are some training options, e.g., number of encoder layers, encoder dimension, decoder dimension, number of warmup steps etc, that are not passed from the commandline. They are pre-configured by the function `get_params()` in pruned_transducer_stateless7_streaming/train.py

You don't need to change these pre-configured parameters. If you really need to change them, please modify `./pruned_transducer_stateless7_streaming/train.py` directly.

Training logs

Training logs and checkpoints are saved in `--exp-dir` (e.g. `pruned_transducer_stateless7_streaming/exp`. You will find the following files in that directory:

- `epoch-1.pt`, `epoch-2.pt`, …

  These are checkpoint files saved at the end of each epoch, containing model `state_dict` and optimizer `state_dict`. To resume training from some checkpoint, say `epoch-10.pt`, you can use:

  ```
  $ ./pruned_transducer_stateless7_streaming/train.py --start-epoch 11
  ```

- `checkpoint-436000.pt`, `checkpoint-438000.pt`, …

  These are checkpoint files saved every `--save-every-n` batches, containing model `state_dict` and optimizer `state_dict`. To resume training from some checkpoint, say `checkpoint-436000`, you can use:

  ```
  $ ./pruned_transducer_stateless7_streaming/train.py --start-batch␣
  ↪436000
  ```

- `tensorboard/`

  This folder contains tensorBoard logs. Training loss, validation loss, learning rate, etc, are recorded in these logs. You can visualize them by:

  ```
  $ cd pruned_transducer_stateless7_streaming/exp/tensorboard
  $ tensorboard dev upload --logdir . --description "pruned␣
  ↪transducer training for LibriSpeech with icefall"
  ```

Hint: If you don't have access to google, you can use the following command to view the tensorboard log locally:

```
cd pruned_transducer_stateless7_streaming/exp/tensorboard
tensorboard --logdir . --port 6008
```

It will print the following message:

```
Serving TensorBoard on localhost; to expose to the network, use a␣
↪proxy or pass --bind_all
TensorBoard 2.8.0 at http://localhost:6008/ (Press CTRL+C to quit)
```

Now start your browser and go to http://localhost:6008 to view the tensorboard logs.

- `log/log-train-xxxx`

  It is the detailed training log in text format, same as the one you saw printed to the console during training.

Usage example

You can use the following command to start the training using 4 GPUs:

```
export CUDA_VISIBLE_DEVICES="0,1,2,3"
./pruned_transducer_stateless7_streaming/train.py \
  --world-size 4 \
  --num-epochs 30 \
  --start-epoch 1 \
  --use-fp16 1 \
  --exp-dir pruned_transducer_stateless7_streaming/exp \
  --full-libri 1 \
  --max-duration 550
```

Decoding

The decoding part uses checkpoints saved by the training part, so you have to run the training part first.

Hint: There are two kinds of checkpoints:

- (1) `epoch-1.pt`, `epoch-2.pt`, . . ., which are saved at the end of each epoch. You can pass `--epoch` to `pruned_transducer_stateless7_streaming/decode.py` to use them.
- (2) `checkpoints-436000.pt`, `epoch-438000.pt`, . . ., which are saved every `--save-every-n` batches. You can pass `--iter` to `pruned_transducer_stateless7_streaming/decode.py` to use them.

We suggest that you try both types of checkpoints and choose the one that produces the lowest WERs.

Tip: To decode a streaming model, you can use either `simulate streaming decoding` in `decode.py` or `real chunk-wise streaming decoding` in `streaming_decode.py`. The difference between `decode.py` and `streaming_decode.py` is that, `decode.py` processes the whole acoustic frames at one time with masking (i.e. same as training), but `streaming_decode.py` processes the acoustic frames chunk by chunk.

Note: `simulate streaming decoding in decode.py and real chunk-size streaming decoding in streaming_decode.py` should produce almost the same results given the same `--decode-chunk-len`.

Simulate streaming decoding

```
$ cd egs/librispeech/ASR
$ ./pruned_transducer_stateless7_streaming/decode.py --help
```

shows the options for decoding. The following options are important for streaming models:

`--decode-chunk-len`

It is same as in `train.py`, which specifies the chunk size for decoding (in frames before subsampling). The default value is 32 (i.e., 320ms).

The following shows two examples (for the two types of checkpoints):

```
for m in greedy_search fast_beam_search modified_beam_search; do
  for epoch in 30; do
    for avg in 12 11 10 9 8; do
      ./pruned_transducer_stateless7_streaming/decode.py \
        --epoch $epoch \
        --avg $avg \
        --decode-chunk-len 32 \
        --exp-dir pruned_transducer_stateless7_streaming/exp \
        --max-duration 600 \
        --decoding-method $m
    done
  done
done
```

```
for m in greedy_search fast_beam_search modified_beam_search; do
  for iter in 474000; do
    for avg in 8 10 12 14 16 18; do
      ./pruned_transducer_stateless7_streaming/decode.py \
        --iter $iter \
        --avg $avg \
        --decode-chunk-len 32 \
        --exp-dir pruned_transducer_stateless7_streaming/exp \
        --max-duration 600 \
        --decoding-method $m
    done
  done
done
```

Real streaming decoding

```
$ cd egs/librispeech/ASR
$ ./pruned_transducer_stateless7_streaming/streaming_decode.py --help
```

shows the options for decoding. The following options are important for streaming models:

`--decode-chunk-len`

It is same as in `train.py`, which specifies the chunk size for decoding (in frames before subsampling). The default value is 32 (i.e., 320ms). For `real streaming decoding`, we will process `decode-chunk-len` acoustic frames at each time.

`--num-decode-streams`

The number of decoding streams that can be run in parallel (very similar to the `bath size`). For `real streaming decoding`, the batches will be packed dynamically, for example, if the `num-decode-streams` equals to 10, then, sequence 1 to 10 will be decoded at first, after a while, suppose sequence 1 and 2 are done, so, sequence 3 to 12 will be processed parallelly in a batch.

The following shows two examples (for the two types of checkpoints):

```
for m in greedy_search fast_beam_search modified_beam_search; do
  for epoch in 30; do
    for avg in 12 11 10 9 8; do
      ./pruned_transducer_stateless7_streaming/decode.py \
        --epoch $epoch \
        --avg $avg \
        --decode-chunk-len 32 \
        --num-decode-streams 100 \
        --exp-dir pruned_transducer_stateless7_streaming/exp \
        --decoding-method $m
    done
  done
done
```

```
for m in greedy_search fast_beam_search modified_beam_search; do
  for iter in 474000; do
    for avg in 8 10 12 14 16 18; do
      ./pruned_transducer_stateless7_streaming/decode.py \
        --iter $iter \
        --avg $avg \
        --decode-chunk-len 16 \
        --num-decode-streams 100 \
        --exp-dir pruned_transducer_stateless7_streaming/exp \
        --decoding-method $m
    done
  done
done
```

Tip: Supporting decoding methods are as follows:

- `greedy_search` : It takes the symbol with largest posterior probability of each frame as the decoding result.

- `beam_search` : It implements Algorithm 1 in https://arxiv.org/pdf/1211.3711.pdf and espnet/nets/beam_search_transducer.py is used as a reference. Basicly, it keeps topk states for each frame, and expands the kept states with their own contexts to next frame.
- `modified_beam_search` : It implements the same algorithm as `beam_search` above, but it runs in batch mode with `--max-sym-per-frame=1` being hardcoded.
- `fast_beam_search` : It implements graph composition between the output `log_probs` and given `FSAs`. It is hard to describe the details in several lines of texts, you can read our paper in https://arxiv.org/pdf/2211.00484.pdf or our rnnt decode code in k2. `fast_beam_search` can decode with `FSAs` on GPU efficiently.
- `fast_beam_search_LG` : The same as `fast_beam_search` above, `fast_beam_search` uses an trivial graph that has only one state, while `fast_beam_search_LG` uses an LG graph (with N-gram LM).
- `fast_beam_search_nbest` : It produces the decoding results as follows:
  - (1) Use `fast_beam_search` to get a lattice
  - (2) Select `num_paths` paths from the lattice using `k2.random_paths()`
  - (3) Unique the selected paths
  - (4) Intersect the selected paths with the lattice and compute the shortest path from the intersection result
  - (5) The path with the largest score is used as the decoding output.
- `fast_beam_search_nbest_LG` : It implements same logic as `fast_beam_search_nbest`, the only difference is that it uses `fast_beam_search_LG` to generate the lattice.

Note: The supporting decoding methods in `streaming_decode.py` might be less than that in `decode.py`, if needed, you can implement them by yourself or file a issue in icefall .

Export Model

Currently it supports exporting checkpoints from `pruned_transducer_stateless7_streaming/exp` in the following ways.

Export `model.state_dict()`

Checkpoints saved by `pruned_transducer_stateless7_streaming/train.py` also include `optimizer.state_dict()`. It is useful for resuming training. But after training, we are interested only in `model.state_dict()`. You can use the following command to extract `model.state_dict()`.

```
# Assume that --epoch 30 --avg 9 produces the smallest WER
# (You can get such information after running ./pruned_transducer_stateless7_streaming/
↪decode.py)

epoch=30
avg=9

./pruned_transducer_stateless7_streaming/export.py \
  --exp-dir ./pruned_transducer_stateless7_streaming/exp \
  --bpe-model data/lang_bpe_500/bpe.model \
  --epoch $epoch \
```

(continues on next page)

(continued from previous page)

```
  --avg  $avg \
  --use-averaged-model=True \
  --decode-chunk-len 32
```

It will generate a file ./pruned_transducer_stateless7_streaming/exp/pretrained.pt.

Hint: To use the generated pretrained.pt for pruned_transducer_stateless7_streaming/decode.py, you can run:

```
cd pruned_transducer_stateless7_streaming/exp
ln -s pretrained.pt epoch-999.pt
```

And then pass --epoch 999 --avg 1 --use-averaged-model 0 to ./pruned_transducer_stateless7_streaming/decode.py.

To use the exported model with ./pruned_transducer_stateless7_streaming/pretrained.py, you can run:

```
./pruned_transducer_stateless7_streaming/pretrained.py \
  --checkpoint ./pruned_transducer_stateless7_streaming/exp/pretrained.pt \
  --bpe-model ./data/lang_bpe_500/bpe.model \
  --method greedy_search \
  --decode-chunk-len 32 \
  /path/to/foo.wav \
  /path/to/bar.wav
```

Export model using torch.jit.script()

```
./pruned_transducer_stateless7_streaming/export.py \
  --exp-dir ./pruned_transducer_stateless7_streaming/exp \
  --bpe-model data/lang_bpe_500/bpe.model \
  --epoch 30 \
  --avg 9 \
  --decode-chunk-len 32 \
  --jit 1
```

Caution: --decode-chunk-len is required to export a ScriptModule.

It will generate a file cpu_jit.pt in the given exp_dir. You can later load it by torch.jit.load("cpu_jit.pt").

Note cpu in the name cpu_jit.pt means the parameters when loaded into Python are on CPU. You can use to("cuda") to move them to a CUDA device.

Export model using `torch.jit.trace()`

```
epoch=30
avg=9

./pruned_transducer_stateless7_streaming/jit_trace_export.py \
  --bpe-model data/lang_bpe_500/bpe.model \
  --use-averaged-model=True \
  --decode-chunk-len 32 \
  --exp-dir ./pruned_transducer_stateless7_streaming/exp \
  --epoch $epoch \
  --avg $avg
```

Caution: `--decode-chunk-len` is required to export a ScriptModule.

It will generate 3 files:

- `./pruned_transducer_stateless7_streaming/exp/encoder_jit_trace.pt`
- `./pruned_transducer_stateless7_streaming/exp/decoder_jit_trace.pt`
- `./pruned_transducer_stateless7_streaming/exp/joiner_jit_trace.pt`

To use the generated files with `./pruned_transducer_stateless7_streaming/jit_trace_pretrained.py`:

```
./pruned_transducer_stateless7_streaming/jit_trace_pretrained.py \
  --encoder-model-filename ./pruned_transducer_stateless7_streaming/exp/encoder_jit_
↪trace.pt \
  --decoder-model-filename ./pruned_transducer_stateless7_streaming/exp/decoder_jit_
↪trace.pt \
  --joiner-model-filename ./pruned_transducer_stateless7_streaming/exp/joiner_jit_trace.
↪pt \
  --bpe-model ./data/lang_bpe_500/bpe.model \
  --decode-chunk-len 32 \
  /path/to/foo.wav
```

Download pretrained models

If you don't want to train from scratch, you can download the pretrained models by visiting the following links:

- pruned_transducer_stateless7_streaming

  See https://github.com/k2-fsa/icefall/blob/master/egs/librispeech/ASR/RESULTS.md for the details of the above pretrained models

Deploy with Sherpa

Please see https://k2-fsa.github.io/sherpa/python/streaming_asr/conformer/index.html# for how to deploy the models in sherpa.

6.3 RNN-LM

6.3.1 Train an RNN langugage model

If you have enough text data, you can train a neural network language model (NNLM) to improve the WER of your E2E ASR system. This tutorial shows you how to train an RNNLM from scratch.

Hint: For how to use an NNLM during decoding, please refer to the following tutorials: *Shallow fusion for Transducer*, *LODR for RNN Transducer*, *LM rescoring for Transducer*

Note: This tutorial is based on the LibriSpeech recipe. Please check it out for the necessary python scripts for this tutorial. We use the LibriSpeech LM-corpus as the LM training set for illustration purpose. You can also collect your own data. The data format is quite simple: each line should contain a complete sentence, and words should be separated by space.

First, let's download the training data for the RNNLM. This can be done via the following command:

```
$ wget https://www.openslr.org/resources/11/librispeech-lm-norm.txt.gz
$ gzip -d librispeech-lm-norm.txt.gz
```

As we are training a BPE-level RNNLM, we need to tokenize the training text, which requires a BPE tokenizer. This can be achieved by executing the following command:

```
$ # if you don't have the BPE
$ GIT_LFS_SKIP_SMUDGE=1 git clone https://huggingface.co/Zengwei/icefall-asr-librispeech-
↪zipformer-2023-05-15
$ cd icefall-asr-librispeech-zipformer-2023-05-15/data/lang_bpe_500
$ git lfs pull --include bpe.model
$ cd ../../..

$ ./local/prepare_lm_training_data.py \
    --bpe-model icefall-asr-librispeech-zipformer-2023-05-15/data/lang_bpe_500/bpe.model␣
↪\
    --lm-data librispeech-lm-norm.txt \
    --lm-archive data/lang_bpe_500/lm_data.pt
```

Now, you should have a file name `lm_data.pt` file store under the directory `data/lang_bpe_500`. This is the packed training data for the RNNLM. We then sort the training data according to its sentence length.

```
$ # This could take a while (~ 20 minutes), feel free to grab a cup of coffee :)
$ ./local/sort_lm_training_data.py \
    --in-lm-data data/lang_bpe_500/lm_data.pt \
    --out-lm-data data/lang_bpe_500/sorted_lm_data.pt \
    --out-statistics data/lang_bpe_500/lm_data_stats.txt
```

The aforementioned steps can be repeated to create a a validation set for you RNNLM. Let's say you have a validation set in `valid.txt`, you can just set `--lm-data valid.txt` and `--lm-archive data/lang_bpe_500/lm-data-valid.pt` when calling `./local/prepare_lm_training_data.py`.

After completing the previous steps, the training and testing sets for training RNNLM are ready. The next step is to train the RNNLM model. The training command is as follows:

```
$ # assume you are in the icefall root directory
$ cd rnn_lm
$ ln -s ../../egs/librispeech/ASR/data .
$ cd ..
$ ./rnn_lm/train.py \
    --world-size 4 \
    --exp-dir ./rnn_lm/exp \
    --start-epoch 0 \
    --num-epochs 10 \
    --use-fp16 0 \
    --tie-weights 1 \
    --embedding-dim 2048 \
    --hidden_dim 2048 \
    --num-layers 3 \
    --batch-size 300 \
    --lm-data rnn_lm/data/lang_bpe_500/sorted_lm_data.pt \
    --lm-data-valid rnn_lm/data/lang_bpe_500/sorted_lm_data.pt
```

Note: You can adjust the RNNLM hyper parameters to control the size of the RNNLM, such as embedding dimension and hidden state dimension. For more details, please run `./rnn_lm/train.py --help`.

Note: The training of RNNLM can take a long time (usually a couple of days).

CHAPTER

SEVEN

CONTRIBUTING

Contributions to `icefall` are very welcomed. There are many possible ways to make contributions and two of them are:

- To write documentation
- To write code
  - (1) To follow the code style in the repository
  - (2) To write a new recipe

In this page, we describe how to contribute documentation and code to `icefall`.

7.1 Contributing to Documentation

We use sphinx for documentation.

Before writing documentation, you have to prepare the environment:

```
$ cd docs
$ pip install -r requirements.txt
```

After setting up the environment, you are ready to write documentation. Please refer to reStructuredText Primer if you are not familiar with `reStructuredText`.

After writing some documentation, you can build the documentation **locally** to preview what it looks like if it is published:

```
$ cd docs
$ make html
```

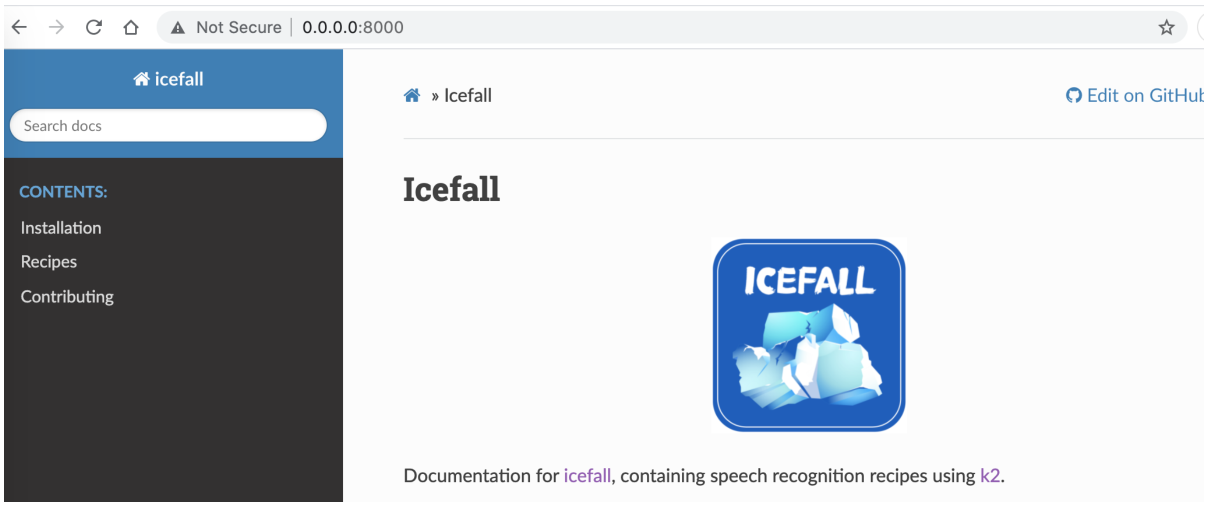
The generated documentation is in `docs/build/html` and can be viewed with the following commands:

```
$ cd docs/build/html
$ python3 -m http.server
```

It will print:

```
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
```

Open your browser, go to http://0.0.0.0:8000/, and you will see the following:



Icefall

Documentation for icefall, containing speech recognition recipes using k2.

Fig. 7.1: View generated documentation locally with `python3 -m http.server`.

7.2 Follow the code style

We use the following tools to make the code style to be as consistent as possible:

- black, to format the code
- flake8, to check the style and quality of the code
- isort, to sort `imports`

The following versions of the above tools are used:

- `black == 22.3.0`
- `flake8 == 5.0.4`
- `isort == 5.10.1`

After running the following commands:

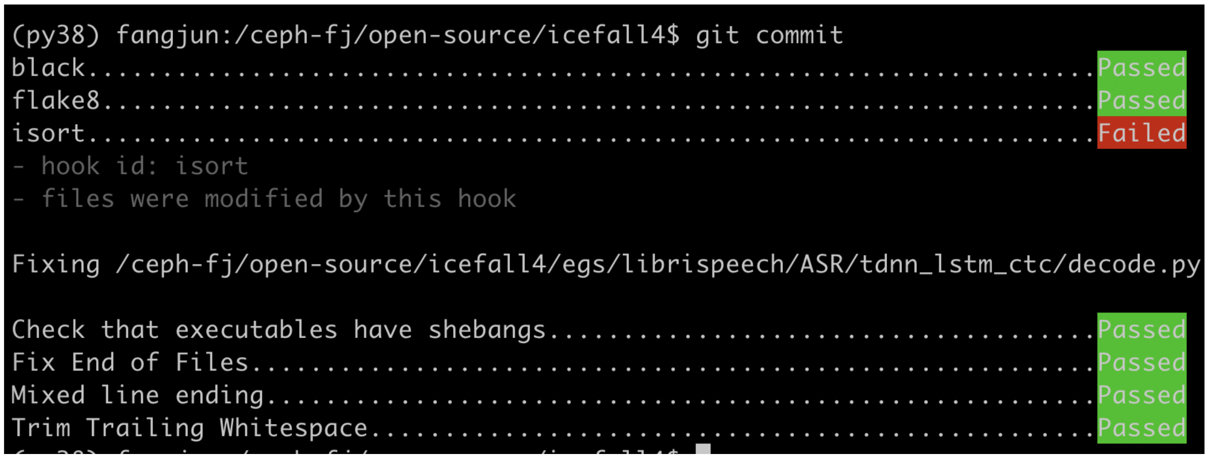
```
$ git clone https://github.com/k2-fsa/icefall
$ cd icefall
$ pip install pre-commit
$ pre-commit install
```

it will run the following checks whenever you run `git commit`, **automatically**:

If any of the above checks failed, your `git commit` was not successful. Please fix any issues reported by the check tools.

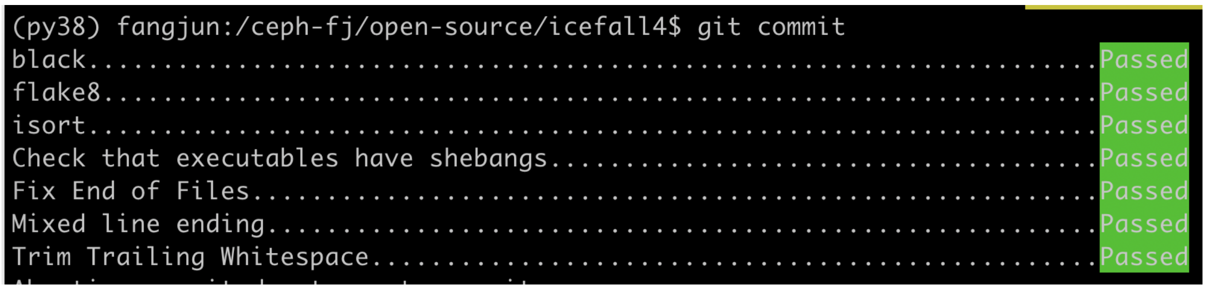
Hint: Some of the check tools, i.e., `black` and `isort` will modify the files to be commited **in-place**. So please run `git status` after failure to see which file has been modified by the tools before you make any further changes.

After fixing all the failures, run `git commit` again and it should succeed this time:



```
(py38) fangjun:/ceph-fj/open-source/icefall4$ git commit
black....................................................................Passed
flake8...................................................................Passed
isort....................................................................Failed
- hook id: isort
- files were modified by this hook

Fixing /ceph-fj/open-source/icefall4/egs/librispeech/ASR/tdnn_lstm_ctc/decode.py

Check that executables have shebangs.....................................Passed
Fix End of Files.........................................................Passed
Mixed line ending........................................................Passed
Trim Trailing Whitespace.................................................Passed
```

Fig. 7.2: pre-commit hooks invoked by `git commit` (Failed).



```
(py38) fangjun:/ceph-fj/open-source/icefall4$ git commit
black....................................................................Passed
flake8...................................................................Passed
isort....................................................................Passed
Check that executables have shebangs.....................................Passed
Fix End of Files.........................................................Passed
Mixed line ending........................................................Passed
Trim Trailing Whitespace.................................................Passed
```

Fig. 7.3: pre-commit hooks invoked by `git commit` (Succeeded).

If you want to check the style of your code before `git commit`, you can do the following:

```
$ pre-commit install
$ pre-commit run
```

Or without installing the pre-commit hooks:

```
$ cd icefall
$ pip install black==22.3.0 flake8==5.0.4 isort==5.10.1
$ black --check your_changed_file.py
$ black your_changed_file.py  # modify it in-place
$
$ flake8 your_changed_file.py
$
$ isort --check your_changed_file.py  # modify it in-place
$ isort your_changed_file.py
```

7.3 How to create a recipe

Hint: Please read *Follow the code style* to adjust your code sytle.

Caution: `icefall` is designed to be as Pythonic as possible. Please use Python in your recipe if possible.

7.3.1 Data Preparation

We recommend you to prepare your training/test/validate dataset with lhotse.

Please refer to https://lhotse.readthedocs.io/en/latest/index.html for how to create a recipe in `lhotse`.

Hint: The `yesno` recipe in `lhotse` is a very good example.

Please refer to https://github.com/lhotse-speech/lhotse/pull/380, which shows how to add a new recipe to `lhotse`.

Suppose you would like to add a recipe for a dataset named `foo`. You can do the following:

```
$ cd egs
$ mkdir -p foo/ASR
$ cd foo/ASR
$ touch prepare.sh
$ chmod +x prepare.sh
```

If your dataset is very simple, please follow egs/yesno/ASR/prepare.sh to write your own `prepare.sh`. Otherwise, please refer to egs/librispeech/ASR/prepare.sh to prepare your data.

7.3.2 Training

Assume you have a fancy model, called bar for the foo recipe, you can organize your files in the following way:

```
$ cd egs/foo/ASR
$ mkdir bar
$ cd bar
$ touch README.md model.py train.py decode.py asr_datamodule.py pretrained.py
```

For instance , the yesno recipe has a tdnn model and its directory structure looks like the following:

```
egs/yesno/ASR/tdnn/
|-- README.md
|-- asr_datamodule.py
|-- decode.py
|-- model.py
|-- pretrained.py
`-- train.py
```

File description:

- README.md

  It contains information of this recipe, e.g., how to run it, what the WER is, etc.

- asr_datamodule.py

  It provides code to create PyTorch dataloaders with train/test/validation dataset.

- decode.py

  It takes as inputs the checkpoints saved during the training stage to decode the test dataset(s).

- model.py

  It contains the definition of your fancy neural network model.

- pretrained.py

  We can use this script to do inference with a pre-trained model.

- train.py

  It contains training code.

Hint: Please take a look at

- egs/yesno/tdnn
- egs/librispeech/tdnn_lstm_ctc
- egs/librispeech/conformer_ctc

to get a feel what the resulting files look like.

Note: Every model in a recipe is kept to be as self-contained as possible. We tolerate duplicate code among different recipes.

The training stage should be invocable by:

```
$ cd egs/foo/ASR
$ ./bar/train.py
$ ./bar/train.py --help
```

7.3.3 Decoding

Please refer to

- https://github.com/k2-fsa/icefall/blob/master/egs/librispeech/ASR/conformer_ctc/decode.py

  If your model is transformer/conformer based.

- https://github.com/k2-fsa/icefall/blob/master/egs/librispeech/ASR/tdnn_lstm_ctc/decode.py

  If your model is TDNN/LSTM based, i.e., there is no attention decoder.

- https://github.com/k2-fsa/icefall/blob/master/egs/yesno/ASR/tdnn/decode.py

  If there is no LM rescoring.

The decoding stage should be invocable by:

```
$ cd egs/foo/ASR
$ ./bar/decode.py
$ ./bar/decode.py --help
```

7.3.4 Pre-trained model

Please demonstrate how to use your model for inference in `egs/foo/ASR/bar/pretrained.py`. If possible, please consider creating a Colab notebook to show that.

CHAPTER

EIGHT

HUGGINGFACE

This section describes how to find pre-trained models. It also demonstrates how to try them from within your browser without installing anything by using Huggingface spaces.

8.1 Pre-trained models

We have uploaded pre-trained models for all recipes in `icefall` to https://huggingface.co/.

You can find them by visiting the following link:

https://huggingface.co/models?search=icefall.

You can also find links of pre-trained models for a specific recipe by looking at the corresponding `RESULTS.md`. For instance:

- https://github.com/k2-fsa/icefall/blob/master/egs/librispeech/ASR/RESULTS.md
- https://github.com/k2-fsa/icefall/blob/master/egs/aishell/ASR/RESULTS.md
- https://github.com/k2-fsa/icefall/blob/master/egs/gigaspeech/ASR/RESULTS.md
- https://github.com/k2-fsa/icefall/blob/master/egs/wenetspeech/ASR/RESULTS.md

8.2 Huggingface spaces

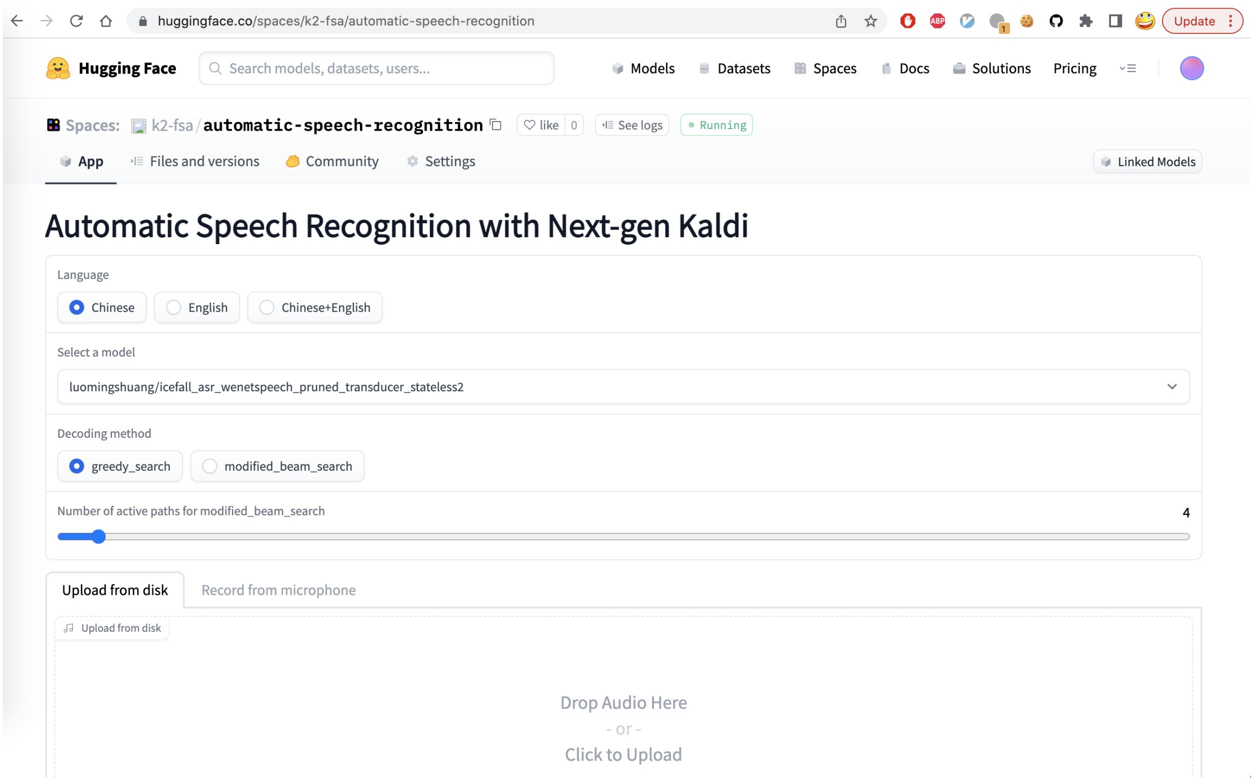
We have integrated the server framework sherpa with Huggingface spaces so that you can try pre-trained models from within your browser without the need to download or install anything.

All you need is a browser, which can be run on Windows, macOS, Linux, or even on your iPad and your phone.

Start your browser and visit the following address:

https://huggingface.co/spaces/k2-fsa/automatic-speech-recognition

and you will see a page like the following screenshot:


huggingface.co/spaces/k2-fsa/automatic-speech-recognition

Spaces: k2-fsa / automatic-speech-recognition

Automatic Speech Recognition with Next-gen Kaldi

Language: [✓ Chinese] English Chinese+English

Select a model: luomingshuang/icefall_asr_wenetspeech_pruned_transducer_stateless2

Decoding method: [✓ greedy_search] modified_beam_search

Number of active paths for modified_beam_search: 4

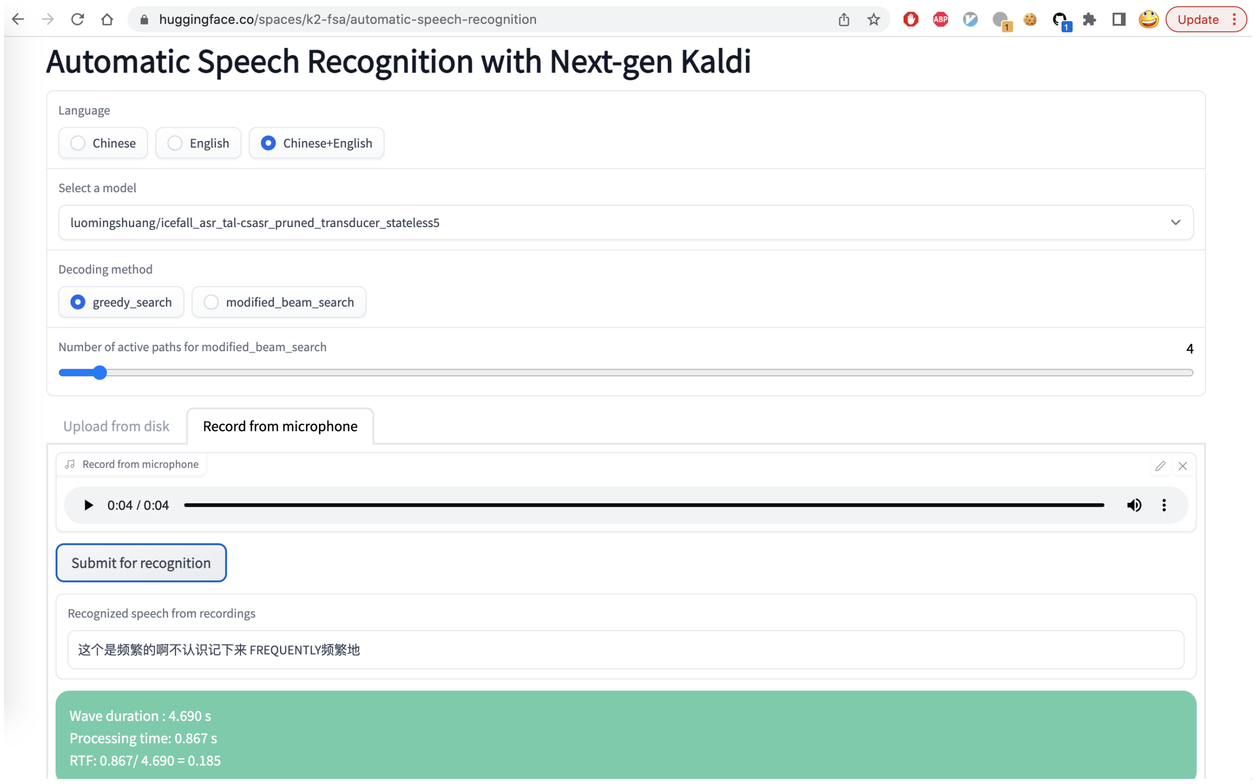
Upload from disk | Record from microphone

Drop Audio Here - or - Click to Upload

You can:

1. Select a language for recognition. Currently, we provide pre-trained models from `icefall` for the following languages: `Chinese`, `English`, and `Chinese+English`.
2. After selecting the target language, you can select a pre-trained model corresponding to the language.
3. Select the decoding method. Currently, it provides `greedy search` and `modified_beam_search`.
4. If you selected `modified_beam_search`, you can choose the number of active paths during the search.
5. Either upload a file or record your speech for recognition.
6. Click the button `Submit for recognition`.
7. Wait for a moment and you will get the recognition results.

The following screenshot shows an example when selecting `Chinese+English`:



Automatic Speech Recognition with Next-gen Kaldi

Language: Chinese / English / [✓ Chinese+English]

Select a model: luomingshuang/icefall_asr_tal-csasr_pruned_transducer_stateless5

Decoding method: [✓ greedy_search] / modified_beam_search

Number of active paths for modified_beam_search: 4

Upload from disk | Record from microphone

Record from microphone

0:04 / 0:04

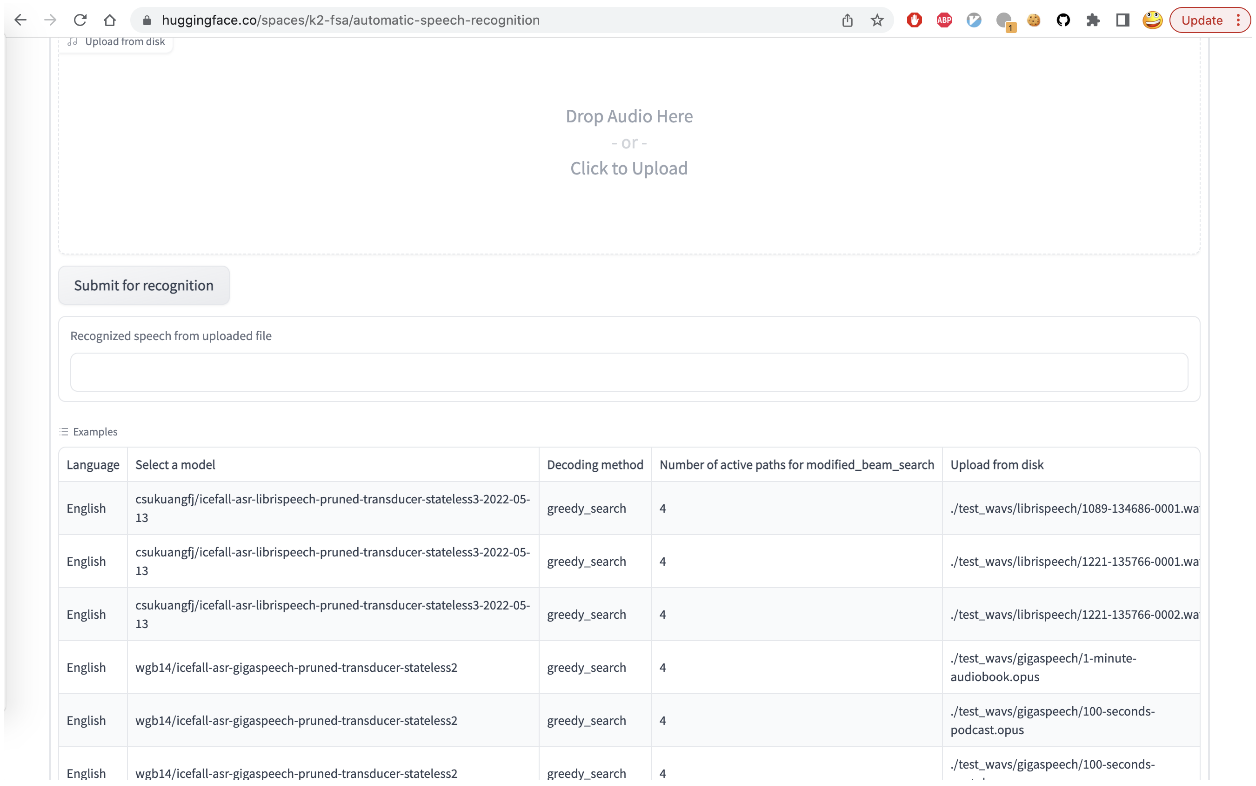
Submit for recognition

Recognized speech from recordings

这个是频繁的啊不认识记下来 FREQUENTLY频繁地

Wave duration : 4.690 s
Processing time: 0.867 s
RTF: 0.867/ 4.690 = 0.185

In the bottom part of the page, you can find a table of examples. You can click one of them and then click `Submit for recognition`.



Submit for recognition

Recognized speech from uploaded file

Examples

Language	Select a model	Decoding method	Number of active paths for modified_beam_search	Upload from disk
English	csukuangfj/icefall-asr-librispeech-pruned-transducer-stateless3-2022-05-13	greedy_search	4	./test_wavs/librispeech/1089-134686-0001.wa
English	csukuangfj/icefall-asr-librispeech-pruned-transducer-stateless3-2022-05-13	greedy_search	4	./test_wavs/librispeech/1221-135766-0001.wa
English	csukuangfj/icefall-asr-librispeech-pruned-transducer-stateless3-2022-05-13	greedy_search	4	./test_wavs/librispeech/1221-135766-0002.wa
English	wgb14/icefall-asr-gigaspeech-pruned-transducer-stateless2	greedy_search	4	./test_wavs/gigaspeech/1-minute-audiobook.opus
English	wgb14/icefall-asr-gigaspeech-pruned-transducer-stateless2	greedy_search	4	./test_wavs/gigaspeech/100-seconds-podcast.opus
English	wgb14/icefall-asr-gigaspeech-pruned-transducer-stateless2	greedy_search	4	./test_wavs/gigaspeech/100-seconds-

8.2.1 YouTube Video

We provide the following YouTube video demonstrating how to use https://huggingface.co/spaces/k2-fsa/automatic-speech-recognition.

Note: To get the latest news of next-gen Kaldi, please subscribe the following YouTube channel by Nadira Povey:

https://www.youtube.com/channel/UC_VaumpkmINz1pNkFXAN9mw

https://youtu.be/ElN3r9dkKE4

CHAPTER

NINE

DECODING WITH LANGUAGE MODELS

This section describes how to use external langugage models during decoding to improve the WER of transducer models. To train an external language model, please refer to this tutorial: *Train an RNN langugage model*.

The following decoding methods with external langugage models are available:

Decoding method	beam=4
`modified_beam_search`	Beam search (i.e. really n-best decoding, the "beam" is the value of n), similar to the original RNN-T paper. Note, this method does not use language model.
`modified_beam_search_lm_shallow_fusion`	As `modified_beam_search`, but interpolate RNN-T scores with language model scores, also known as shallow fusion
`modified_beam_search_LODR`	As `modified_beam_search_lm_shallow_fusion`, but subtract score of a (BPE-symbol-level) bigram backoff language model used as an approximation to the internal language model of RNN-T.
`modified_beam_search_lm_rescore`	As `modified_beam_search`, but rescore the n-best hypotheses with external language model (e.g. RNNLM) and re-rank them.
`modified_beam_search_lm_rescore_LODR`	As `modified_beam_search_lm_rescore`, but also subtract the score of a (BPE-symbol-level) bigram backoff language model during re-ranking.

9.1 Shallow fusion for Transducer

External language models (LM) are commonly used to improve WERs for E2E ASR models. This tutorial shows you how to perform `shallow fusion` with an external LM to improve the word-error-rate of a transducer model.

Note: This tutorial is based on the recipe pruned_transducer_stateless7_streaming, which is a streaming transducer model trained on LibriSpeech. However, you can easily apply shallow fusion to other recipes. If you encounter any problems, please open an issue here icefall.

Note: For simplicity, the training and testing corpus in this tutorial is the same (LibriSpeech). However, you can change the testing set to any other domains (e.g GigaSpeech) and use an external LM trained on that domain.

Hint: We recommend you to use a GPU for decoding.

For illustration purpose, we will use a pre-trained ASR model from this link. If you want to train your model from scratch, please have a look at *Pruned transducer statelessX*.

As the initial step, let's download the pre-trained model.

```
$ GIT_LFS_SKIP_SMUDGE=1 git clone https://huggingface.co/Zengwei/icefall-asr-librispeech-
↪pruned-transducer-stateless7-streaming-2022-12-29
$ cd icefall-asr-librispeech-pruned-transducer-stateless7-streaming-2022-12-29/exp
$ git lfs pull --include "pretrained.pt"
$ ln -s pretrained.pt epoch-99.pt # create a symbolic link so that the checkpoint can be␣
↪loaded
$ cd ../data/lang_bpe_500
$ git lfs pull --include bpe.model
$ cd ../../..
```

To test the model, let's have a look at the decoding results without using LM. This can be done via the following command:

```
$ exp_dir=./icefall-asr-librispeech-pruned-transducer-stateless7-streaming-2022-12-29/
↪exp/
$ ./pruned_transducer_stateless7_streaming/decode.py \
    --epoch 99 \
    --avg 1 \
    --use-averaged-model False \
    --exp-dir $exp_dir \
    --bpe-model ./icefall-asr-librispeech-pruned-transducer-stateless7-streaming-2022-12-
↪29/data/lang_bpe_500/bpe.model \
    --max-duration 600 \
    --decode-chunk-len 32 \
    --decoding-method modified_beam_search
```

The following WERs are achieved on test-clean and test-other:

```
$ For test-clean, WER of different settings are:
$ beam_size_4       3.11    best for test-clean
$ For test-other, WER of different settings are:
$ beam_size_4       7.93    best for test-other
```

These are already good numbers! But we can further improve it by using shallow fusion with external LM. Training a language model usually takes a long time, we can download a pre-trained LM from this link.

```
$ # download the external LM
$ GIT_LFS_SKIP_SMUDGE=1 git clone https://huggingface.co/ezerhouni/icefall-librispeech-
↪rnn-lm
$ # create a symbolic link so that the checkpoint can be loaded
$ pushd icefall-librispeech-rnn-lm/exp
$ git lfs pull --include "pretrained.pt"
$ ln -s pretrained.pt epoch-99.pt
$ popd
```

Note: This is an RNN LM trained on the LibriSpeech text corpus. So it might not be ideal for other corpus. You may also train a RNN LM from scratch. Please refer to this script for training a RNN LM and this script to train a transformer LM.

To use shallow fusion for decoding, we can execute the following command:

```
$ exp_dir=./icefall-asr-librispeech-pruned-transducer-stateless7-streaming-2022-12-29/exp
$ lm_dir=./icefall-librispeech-rnn-lm/exp
$ lm_scale=0.29
$ ./pruned_transducer_stateless7_streaming/decode.py \
    --epoch 99 \
    --avg 1 \
    --use-averaged-model False \
    --beam-size 4 \
    --exp-dir $exp_dir \
    --max-duration 600 \
    --decode-chunk-len 32 \
    --decoding-method modified_beam_search_lm_shallow_fusion \
    --bpe-model ./icefall-asr-librispeech-pruned-transducer-stateless7-streaming-2022-12-
↪29/data/lang_bpe_500/bpe.model \
    --use-shallow-fusion 1 \
    --lm-type rnn \
    --lm-exp-dir $lm_dir \
    --lm-epoch 99 \
    --lm-scale $lm_scale \
    --lm-avg 1 \
    --rnn-lm-embedding-dim 2048 \
    --rnn-lm-hidden-dim 2048 \
    --rnn-lm-num-layers 3 \
    --lm-vocab-size 500
```

Note that we set `--decoding-method modified_beam_search_lm_shallow_fusion` and `--use-shallow-fusion True` to use shallow fusion. `--lm-type` specifies the type of neural LM we are going to use, you can either choose between `rnn` or `transformer`. The following three arguments are associated with the rnn:

- **`--rnn-lm-embedding-dim`**
  The embedding dimension of the RNN LM
- **`--rnn-lm-hidden-dim`**
  The hidden dimension of the RNN LM
- **`--rnn-lm-num-layers`**
  The number of RNN layers in the RNN LM.

The decoding result obtained with the above command are shown below.

```
$ For test-clean, WER of different settings are:
$ beam_size_4       2.77    best for test-clean
$ For test-other, WER of different settings are:
$ beam_size_4       7.08    best for test-other
```

The improvement of shallow fusion is very obvious! The relative WER reduction on test-other is around 10.5%. A few parameters can be tuned to further boost the performance of shallow fusion:

- `--lm-scale`

  Controls the scale of the LM. If too small, the external language model may not be fully utilized; if too large, the LM score may dominant during decoding, leading to bad WER. A typical value of this is around 0.3.

- `--beam-size`

The number of active paths in the search beam. It controls the trade-off between decoding efficiency and accuracy.

Here, we also show how *–beam-size* effect the WER and decoding time:

Table 9.1: WERs and decoding time (on test-clean) of shallow fusion with different beam sizes

Beam size	test-clean	test-other	Decoding time on test-clean (s)
4	2.77	7.08	262
8	2.62	6.65	352
12	2.58	6.65	488

As we see, a larger beam size during shallow fusion improves the WER, but is also slower.

9.2 LODR for RNN Transducer

As a type of E2E model, neural transducers are usually considered as having an internal language model, which learns the language level information on the training corpus. In real-life scenario, there is often a mismatch between the training corpus and the target corpus space. This mismatch can be a problem when decoding for neural transducer models with language models as its internal language can act "against" the external LM. In this tutorial, we show how to use Low-order Density Ratio to alleviate this effect to further improve the performance of langugae model integration.

Note: This tutorial is based on the recipe pruned_transducer_stateless7_streaming, which is a streaming transducer model trained on LibriSpeech. However, you can easily apply LODR to other recipes. If you encounter any problems, please open an issue here icefall.

Note: For simplicity, the training and testing corpus in this tutorial are the same (LibriSpeech). However, you can change the testing set to any other domains (e.g GigaSpeech) and prepare the language models using that corpus.

First, let's have a look at some background information. As the predecessor of LODR, Density Ratio (DR) is first proposed here to address the language information mismatch between the training corpus (source domain) and the testing corpus (target domain). Assuming that the source domain and the test domain are acoustically similar, DR derives the following formular for decoding with Bayes' theorem:

$$\text{score}\left(y_u|x,y\right) = \log p\left(y_u|x,y_{1:u-1}\right) + \lambda_1 \log p_{\text{Target LM}}\left(y_u|x,y_{1:u-1}\right) - \lambda_2 \log p_{\text{Source LM}}\left(y_u|x,y_{1:u-1}\right)$$

where λ_1 and λ_2 are the weights of LM scores for target domain and source domain respectively. Here, the source domain LM is trained on the training corpus. The only difference in the above formular compared to shallow fusion is the subtraction of the source domain LM.

Some works treat the predictor and the joiner of the neural transducer as its internal LM. However, the LM is considered to be weak and can only capture low-level language information. Therefore, LODR proposed to use a low-order n-gram LM as an approximation of the ILM of the neural transducer. This leads to the following formula during decoding for transducer model:

$$\text{score}\left(y_u|x,y\right) = \log p_{rnnt}\left(y_u|x,y_{1:u-1}\right) + \lambda_1 \log p_{\text{Target LM}}\left(y_u|x,y_{1:u-1}\right) - \lambda_2 \log p_{\text{bi-gram}}\left(y_u|x,y_{1:u-1}\right)$$

In LODR, an additional bi-gram LM estimated on the source domain (e.g training corpus) is required. Comared to DR, the only difference lies in the choice of source domain LM. According to the original paper, LODR achieves similar performance compared DR in both intra-domain and cross-domain settings. As a bi-gram is much faster to evaluate, LODR is usually much faster.

Now, we will show you how to use LODR in `icefall`. For illustration purpose, we will use a pre-trained ASR model from this link. If you want to train your model from scratch, please have a look at *Pruned transducer statelessX*. The testing scenario here is intra-domain (we decode the model trained on LibriSpeech on LibriSpeech testing sets).

As the initial step, let's download the pre-trained model.

```
$ GIT_LFS_SKIP_SMUDGE=1 git clone https://huggingface.co/Zengwei/icefall-asr-librispeech-
↪pruned-transducer-stateless7-streaming-2022-12-29
$ cd icefall-asr-librispeech-pruned-transducer-stateless7-streaming-2022-12-29/exp
$ git lfs pull --include "pretrained.pt"
$ ln -s pretrained.pt epoch-99.pt # create a symbolic link so that the checkpoint can be␣
↪loaded
$ cd ../data/lang_bpe_500
$ git lfs pull --include bpe.model
$ cd ../../..
```

To test the model, let's have a look at the decoding results **without** using LM. This can be done via the following command:

```
$ exp_dir=./icefall-asr-librispeech-pruned-transducer-stateless7-streaming-2022-12-29/
↪exp/
$ ./pruned_transducer_stateless7_streaming/decode.py \
    --epoch 99 \
    --avg 1 \
    --use-averaged-model False \
    --exp-dir $exp_dir \
    --bpe-model ./icefall-asr-librispeech-pruned-transducer-stateless7-streaming-2022-12-
↪29/data/lang_bpe_500/bpe.model \
    --max-duration 600 \
    --decode-chunk-len 32 \
    --decoding-method modified_beam_search
```

The following WERs are achieved on test-clean and test-other:

```
$ For test-clean, WER of different settings are:
$ beam_size_4       3.11    best for test-clean
$ For test-other, WER of different settings are:
$ beam_size_4       7.93    best for test-other
```

Then, we download the external language model and bi-gram LM that are necessary for LODR. Note that the bi-gram is estimated on the LibriSpeech 960 hours' text.

```
$ # download the external LM
$ GIT_LFS_SKIP_SMUDGE=1 git clone https://huggingface.co/ezerhouni/icefall-librispeech-
↪rnn-lm
$ # create a symbolic link so that the checkpoint can be loaded
$ pushd icefall-librispeech-rnn-lm/exp
$ git lfs pull --include "pretrained.pt"
$ ln -s pretrained.pt epoch-99.pt
$ popd
$
$ # download the bi-gram
```

(continues on next page)

(continued from previous page)

```
$ git lfs install
$ git clone https://huggingface.co/marcoyang/librispeech_bigram
$ pushd data/lang_bpe_500
$ ln -s ../../librispeech_bigram/2gram.fst.txt .
$ popd
```

Then, we perform LODR decoding by setting `--decoding-method` to `modified_beam_search_lm_LODR`:

```
$ exp_dir=./icefall-asr-librispeech-pruned-transducer-stateless7-streaming-2022-12-29/exp
$ lm_dir=./icefall-librispeech-rnn-lm/exp
$ lm_scale=0.42
$ LODR_scale=-0.24
$ ./pruned_transducer_stateless7_streaming/decode.py \
    --epoch 99 \
    --avg 1 \
    --use-averaged-model False \
    --beam-size 4 \
    --exp-dir $exp_dir \
    --max-duration 600 \
    --decode-chunk-len 32 \
    --decoding-method modified_beam_search_LODR \
    --bpe-model ./icefall-asr-librispeech-pruned-transducer-stateless7-streaming-2022-12-
↪29/data/lang_bpe_500/bpe.model \
    --use-shallow-fusion 1 \
    --lm-type rnn \
    --lm-exp-dir $lm_dir \
    --lm-epoch 99 \
    --lm-scale $lm_scale \
    --lm-avg 1 \
    --rnn-lm-embedding-dim 2048 \
    --rnn-lm-hidden-dim 2048 \
    --rnn-lm-num-layers 3 \
    --lm-vocab-size 500 \
    --tokens-ngram 2 \
    --ngram-lm-scale $LODR_scale
```

There are two extra arguments that need to be given when doing LODR. `--tokens-ngram` specifies the order of n-gram. As we are using a bi-gram, we set it to 2. `--ngram-lm-scale` is the scale of the bi-gram, it should be a negative number as we are subtracting the bi-gram's score during decoding.

The decoding results obtained with the above command are shown below:

```
$ For test-clean, WER of different settings are:
$ beam_size_4       2.61    best for test-clean
$ For test-other, WER of different settings are:
$ beam_size_4       6.74    best for test-other
```

Recall that the lowest WER we obtained in *Shallow fusion for Transducer* with beam size of 4 is `2.77/7.08`, LODR indeed **further improves** the WER. We can do even better if we increase `--beam-size`:

Table 9.2: WER of LODR with different beam sizes

Beam size	test-clean	test-other
4	2.61	6.74
8	2.45	6.38
12	2.4	6.23

9.3 LM rescoring for Transducer

LM rescoring is a commonly used approach to incorporate external LM information. Unlike shallow-fusion-based methods (see *Shallow fusion for Transducer*, *LODR for RNN Transducer*), rescoring is usually performed to re-rank the n-best hypotheses after beam search. Rescoring is usually more efficient than shallow fusion since less computation is performed on the external LM. In this tutorial, we will show you how to use external LM to rescore the n-best hypotheses decoded from neural transducer models in icefall.

Note: This tutorial is based on the recipe pruned_transducer_stateless7_streaming, which is a streaming transducer model trained on LibriSpeech. However, you can easily apply shallow fusion to other recipes. If you encounter any problems, please open an issue here.

Note: For simplicity, the training and testing corpus in this tutorial is the same (LibriSpeech). However, you can change the testing set to any other domains (e.g GigaSpeech) and use an external LM trained on that domain.

Hint: We recommend you to use a GPU for decoding.

For illustration purpose, we will use a pre-trained ASR model from this link. If you want to train your model from scratch, please have a look at *Pruned transducer statelessX*.

As the initial step, let's download the pre-trained model.

```
$ GIT_LFS_SKIP_SMUDGE=1 git clone https://huggingface.co/Zengwei/icefall-asr-librispeech-
↪pruned-transducer-stateless7-streaming-2022-12-29
$ cd icefall-asr-librispeech-pruned-transducer-stateless7-streaming-2022-12-29/exp
$ git lfs pull --include "pretrained.pt"
$ ln -s pretrained.pt epoch-99.pt # create a symbolic link so that the checkpoint can be␣
↪loaded
$ cd ../data/lang_bpe_500
$ git lfs pull --include bpe.model
$ cd ../../..
```

As usual, we first test the model's performance without external LM. This can be done via the following command:

```
$ exp_dir=./icefall-asr-librispeech-pruned-transducer-stateless7-streaming-2022-12-29/
↪exp/
$ ./pruned_transducer_stateless7_streaming/decode.py \
    --epoch 99 \
    --avg 1 \
    --use-averaged-model False \
    --exp-dir $exp_dir \
```

(continues on next page)

(continued from previous page)

```
    --bpe-model ./icefall-asr-librispeech-pruned-transducer-stateless7-streaming-2022-12-
↪29/data/lang_bpe_500/bpe.model \
    --max-duration 600 \
    --decode-chunk-len 32 \
    --decoding-method modified_beam_search
```

The following WERs are achieved on test-clean and test-other:

```
$ For test-clean, WER of different settings are:
$ beam_size_4       3.11    best for test-clean
$ For test-other, WER of different settings are:
$ beam_size_4       7.93    best for test-other
```

Now, we will try to improve the above WER numbers via external LM rescoring. We will download a pre-trained LM from this link.

Note: This is an RNN LM trained on the LibriSpeech text corpus. So it might not be ideal for other corpus. You may also train a RNN LM from scratch. Please refer to this script for training a RNN LM and this script to train a transformer LM.

```
$ # download the external LM
$ GIT_LFS_SKIP_SMUDGE=1 git clone https://huggingface.co/ezerhouni/icefall-librispeech-
↪rnn-lm
$ # create a symbolic link so that the checkpoint can be loaded
$ pushd icefall-librispeech-rnn-lm/exp
$ git lfs pull --include "pretrained.pt"
$ ln -s pretrained.pt epoch-99.pt
$ popd
```

With the RNNLM available, we can rescore the n-best hypotheses generated from *modified_beam_search*. Here, *n* should be the number of beams, i.e `--beam-size`. The command for LM rescoring is as follows. Note that the `--decoding-method` is set to *modified_beam_search_lm_rescore* and `--use-shallow-fusion` is set to *False*.

```
$ exp_dir=./icefall-asr-librispeech-pruned-transducer-stateless7-streaming-2022-12-29/exp
$ lm_dir=./icefall-librispeech-rnn-lm/exp
$ lm_scale=0.43
$ ./pruned_transducer_stateless7_streaming/decode.py \
    --epoch 99 \
    --avg 1 \
    --use-averaged-model False \
    --beam-size 4 \
    --exp-dir $exp_dir \
    --max-duration 600 \
    --decode-chunk-len 32 \
    --decoding-method modified_beam_search_lm_rescore \
    --bpe-model ./icefall-asr-librispeech-pruned-transducer-stateless7-streaming-2022-12-
↪29/data/lang_bpe_500/bpe.model \
    --use-shallow-fusion 0 \
    --lm-type rnn \
    --lm-exp-dir $lm_dir \
    --lm-epoch 99 \
```

(continues on next page)

(continued from previous page)

```
    --lm-scale $lm_scale \
    --lm-avg 1 \
    --rnn-lm-embedding-dim 2048 \
    --rnn-lm-hidden-dim 2048 \
    --rnn-lm-num-layers 3 \
    --lm-vocab-size 500
```

```
$ For test-clean, WER of different settings are:
$ beam_size_4       2.93    best for test-clean
$ For test-other, WER of different settings are:
$ beam_size_4       7.6     best for test-other
```

Great! We made some improvements! Increasing the size of the n-best hypotheses will further boost the performance, see the following table:

Table 9.3: WERs of LM rescoring with different beam sizes

Beam size	test-clean	test-other
4	2.93	7.6
8	2.67	7.11
12	2.59	6.86

In fact, we can also apply LODR (see *LODR for RNN Transducer*) when doing LM rescoring. To do so, we need to download the bi-gram required by LODR:

```
$ # download the bi-gram
$ git lfs install
$ git clone https://huggingface.co/marcoyang/librispeech_bigram
$ pushd data/lang_bpe_500
$ ln -s ../../librispeech_bigram/2gram.arpa .
$ popd
```

Then we can performn LM rescoring + LODR by changing the decoding method to *modified_beam_search_lm_rescore_LODR*.

Note: This decoding method requires the dependency of kenlm. You can install it via this command: *pip install https://github.com/kpu/kenlm/archive/master.zip*.

```
$ exp_dir=./icefall-asr-librispeech-pruned-transducer-stateless7-streaming-2022-12-29/exp
$ lm_dir=./icefall-librispeech-rnn-lm/exp
$ lm_scale=0.43
$ ./pruned_transducer_stateless7_streaming/decode.py \
    --epoch 99 \
    --avg 1 \
    --use-averaged-model False \
    --beam-size 4 \
    --exp-dir $exp_dir \
    --max-duration 600 \
    --decode-chunk-len 32 \
    --decoding-method modified_beam_search_lm_rescore_LODR \
    --bpe-model ./icefall-asr-librispeech-pruned-transducer-stateless7-streaming-2022-12-
```

(continues on next page)

(continued from previous page)

```
↪29/data/lang_bpe_500/bpe.model \
    --use-shallow-fusion 0 \
    --lm-type rnn \
    --lm-exp-dir $lm_dir \
    --lm-epoch 99 \
    --lm-scale $lm_scale \
    --lm-avg 1 \
    --rnn-lm-embedding-dim 2048 \
    --rnn-lm-hidden-dim 2048 \
    --rnn-lm-num-layers 3 \
    --lm-vocab-size 500
```

You should see the following WERs after executing the commands above:

```
$ For test-clean, WER of different settings are:
$ beam_size_4       2.9     best for test-clean
$ For test-other, WER of different settings are:
$ beam_size_4       7.57    best for test-other
```

It's slightly better than LM rescoring. If we further increase the beam size, we will see further improvements from LM rescoring + LODR:

Table 9.4: WERs of LM rescoring + LODR with different beam sizes

Beam size	test-clean	test-other
4	2.9	7.57
8	2.63	7.04
12	2.52	6.73

As mentioned earlier, LM rescoring is usually faster than shallow-fusion based methods. Here, we benchmark the WERs and decoding speed of them:

Table 9.5: LM-rescoring-based methods vs shallow-fusion-based methods (The numbers in each field is WER on test-clean, WER on test-other and decoding time on test-clean)

Decoding method	beam=4	beam=8	beam=12
`modified_beam_search`	3.11/7.93; 132s	3.1/7.95; 177s	3.1/7.96; 210s
`modified_beam_search_lm_shallow_fusion`	2.77/7.08; 262s	2.62/6.65; 352s	2.58/6.65; 488s
`modified_beam_search_LODR`	2.61/6.74; 400s	2.45/6.38; 610s	2.4/6.23; 870s
`modified_beam_search_lm_rescore`	2.93/7.6; 156s	2.67/7.11; 203s	2.59/6.86; 255s
`modified_beam_search_lm_rescore_LODR`	2.9/7.57; 160s	2.63/7.04; 203s	2.52/6.73; 263s

Note: Decoding is performed with a single 32G V100, we set `--max-duration` to 600. Decoding time here is only for reference and it may vary.