Text-to-speech experiment on ESPnet
Lab 10

[Objective]

1. Learn how to do train a TTS model using ESPnet

[Task description]

ESPnet (https://github.com/espnet/espnet) is an end-to-end speech processing toolkit, mainly focuses on end-to-end speech recognition and end-to-end text-to-speech (TTS). Here, we will go through the basic pipeline of training a classical TTS model, Tacotron2 with ESPnet.

The full installation (https://espnet.github.io/espnet/installation.html) would create a new Python environment. Since we've used ESPnet in Lab8, let's omit the installation and use the Python environments we already have.

The dataset used in this lab is CSMSC, a popular open-source Mandarin TTS dataset (https://www.data-baker.com/open_source.html). For simply showing the training pipeline, we only use a small portion of the full dataset.

[ESPnet installation]

Please ignore this step if you 1) have already cloned ESPnet and 2) have a Python environment containing the espnet package.

1) Cloning ESPnet

git clone https://github.com/espnet/espnet.git

2) Setting up a Python environment

conda create -n env_name

conda activate env_name

pip install espnet chainer

[ESPnet configuration]

1) Making a link to Kaldi

cd espnet/tools

ln -s kaldi_path kaldi

(Though ESPnet uses PyTorch to train models, it requires Kaldi to preprocess and organize data.)

2) Specifying the Python environment

cd espnet/tools

rm -f activate_python.sh && touch activate_python.sh

(Here, we specify nothing, i.e., using an environment manually activated.)

zhuym@sustc.edu.cn

[Script modification]

From now on, all operations are done in the directory *espnet/egs/csmsc/tts1*, which is like Kaldi.

1) **Entering the egs directory**

   cd espnet/egs/csmsc/tts1/

   (All files generated in the training process are placed here, e.g., processed features, saved models.)

2) **Changing the data root**

   Open *run.sh*, find the line:

   db_root=downloads

   and replace it with:

   db_root=/data/cs310/baibing/datasets

   (You can change it to any folder that contains the *CSMSC* dataset.)

3) **Changing the data root**

   Open *local/data_prep.sh*, find the first line:

   #!/usr/bin/env bash -e

   and replace it with:

   #!/usr/bin/env bash

[Training pipeline]

Before running the scripts, make sure to have activated the Python environment.

1) **Stage 0: data preparation**

   ./run.sh --stage 0 --stop_stage 0

   This step makes files containing information of the data used in training, i.e., *wav.scp*, *text*, *utt2spk*, and *spk2utt*. There are 2 more files: *segments* and *utt2dur*, recording the voicing boundaries and durations of all utterances. The 2 extra files are not required but help training.

2) **Stage 1: feature generation**

   ./run.sh --stage 1 --stop_stage 1

   This step extracts *log-mel-spectrograms* from raw audios as input to the TTS model, then normalize these features, and finally splits the data into 3 standard subsets: *train*, *dev*, and *test*.

3) **Stage 2: dictionary and json preparation**

   ./run.sh --stage 2 --stop_stage 2

   This step makes a dictionary of all tokens appearing text transcripts, where each token is assigned with an index. Then 3 *data.json* are generated for the 3 subsets. This is the file directly used in model training, providing 1) input token indices,

and 2) where to load output acoustic features, as show below.


```
{
    "utts": {
        "000001": {
            "input": [
                {
                    "feat": "/data/cs310/baibing/espnet/egs/csmsc/tts1/dump/train_n
o_dev/feats.1.ark:7",
                    "name": "input1",
                    "shape": [
                        170,
                        80
                    ]
                }
            ],
            "output": [
                {
                    "name": "target1",
                    "shape": [
                        15,
                        223
                    ],
                    "text": "k a2 er2 p u3 p ei2 uai4 s uen1 uan2 h ua2 t i1",
                    "token": "k a2 er2 p u3 p ei2 uai4 s uen1 uan2 h ua2 t i1",
                    "tokenid": "129 3 58 148 157 148 41 168 151 186 170 64 161 154
65"
                }
            ],
            "utt2spk": "csmsc"
        },
    }
}
```


4) Stage 4-5: decoding and synthesis

./run.sh --stage 4 --stop_stage 5

The trained model generates *log-mel-spectrograms* of input text in *dev* and *test*. Then the generated *log-mel-spectrograms* are denormalized and converted to waveforms by the Griffin-Lim algorithm.

[Adding personal data]

The above pipeline shows how to train a TTS model with a published dataset. We can also add some data to the training set.

The simplest way is to add lines to files in *data* after *stage 1*. Make sure that each audio clip you want to add is included in all files; otherwise, it would be ignored during validation. You also need to make sure the sampling rate of additional audios are the same as that of original data. If they don't, please resample them using tools like SOX.

Alternatively, you can make a combined dataset of the original one and extra audios, and process the data with the same pipeline.