ESPnet tutorial

0. **Preparation**

```
$ ssh <user name>@login.clsp.jhu.edu
$ ssh bXX
$ mkdir -p /export/<aYY or bYY>/<user name>/<your working directory>
$ cd /export/<aYY or bYY>/<user name>/<your working directory>
```

   - login to some computer node bXX (do not do the experiment at the login nodes)
   - move to the experimental directory on aYY or bYY (do not do the experiment at your home directory)

1. **download ESPnet**

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

2. **set the environment**

   - To use cuda (and cudnn), make sure to set paths in your `.bashrc` or `.bash_profile` appropriately.

```
CUDAROOT=/path/to/cuda

export PATH=$CUDAROOT/bin:$PATH
export LD_LIBRARY_PATH=$CUDAROOT/lib64:$LD_LIBRARY_PATH
export CUDA_HOME=$CUDAROOT
export CUDA_PATH=$CUDAROOT
```

   - (Optional) if you want to use multiple GPUs, you should install nccl and set paths in your `.bashrc` or `.bash_profile` appropriately, for example:

```
CUDAROOT=/path/to/cuda
NCCL_ROOT=/path/to/nccl

export CPATH=$NCCL_ROOT/include:$CPATH
export LD_LIBRARY_PATH=$NCCL_ROOT/lib/:$CUDAROOT/lib64:$LD_LIBRARY_PATH
export LIBRARY_PATH=$NCCL_ROOT/lib/:$LIBRARY_PATH
export CUDA_HOME=$CUDAROOT
export CUDA_PATH=$CUDAROOT
```

- **Easiest way is to use my setup**

```
CUDAROOT=/home/shinji/tools/cuda
NCCL_ROOT=/home/shinji/tools/ncll/nccl_2.1.15-1+cuda8.0_x86_64/

export CPATH=$NCCL_ROOT/include:$CPATH
export LD_LIBRARY_PATH=$NCCL_ROOT/lib/:$CUDAROOT/lib64:$LD_LIBRARY_PATH
export LIBRARY_PATH=$NCCL_ROOT/lib/:$LIBRARY_PATH
export CUDA_HOME=$CUDAROOT
export CUDA_PATH=$CUDAROOT
```

- [ ] **checkpoint 1)**: check whether CUDA (and NCCL) paths are correctly set by

```
$ echo $CUDA_PATH
/path/to/cuda
$ echo $CUDA_HOME
/path/to/cuda
$ echo $LD_LIBRARY_PATH
/path/to/nccl/lib/:/path/to/cuda/lib64:/path/to/nccl/lib/:/path/to/cuda/li
b64:
$ nvcc -V
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2017 NVIDIA Corporation
Built on Fri_Nov__3_21:07:56_CDT_2017
Cuda compilation tools, release 9.1, V9.1.85
```

3. **Installation**
   - move to the `espnet/tools` directory, and make by specifying your Kaldi directory

```
$ cd espnet/tools
$ make KALDI=/path/to/kaldi
```

   - **Easiest way is to use compiled one**

```
cp -r /export/a08/shinji/201707e2e/espnet_tutorial espnet
```

- [ ] **checkpoint 2)**: check whether pytorch, chainer, and warpctc are correctly installed

```
$ cd espnet/tools
$ . venv/bin/activate
$ python
Python 2.7.13 (default, Nov 24 2017, 17:33:09)
[GCC 6.3.0 20170516] on linux2
Type "help", "copyright", "credits" or "license" for     more information.
>>> import torch
>>> torch.cuda.is_available()
True
>>> import warpctc_pytorch
>>> import chainer
>>> exit()
$ deactivate
```

4. **Run the CMU Census Database (AN4) recipe**
    - move to the recipe directory

```
cd espnet/egs/an4/asr1
```

    - change `cmd.sh` to use the CLSP gridengine

```
# JHU setup
#export train_cmd="queue.pl --mem 2G"
#export cuda_cmd="queue.pl --mem 2G --gpu 1 --config conf/gpu.conf"
#export decode_cmd="queue.pl --mem 4G"
```

->

```
# JHU setup
export train_cmd="queue.pl --mem 2G"
export cuda_cmd="queue.pl --mem 2G --gpu 1 --config conf/gpu.conf"
export decode_cmd="queue.pl --mem 4G"
```

    - [ ] **checkpoint 3)**: check whether `cmd.sh` is correctly set

```
$ . ./cmd.sh
$ echo $train_cmd
queue.pl --mem 2G
$ echo $cuda_cmd
queue.pl --mem 2G --gpu 1 --config conf/gpu.conf
```

- [ ] **checkpoint 4)**: check whether Kaldi is correctly set

```
$ . ./path.sh
(venv) $ copy-feats
copy-feats

Copy features [and possibly change format]
Usage: copy-feats [options] <feature-rspecifier> <feature-wspecifier>
or:   copy-feats [options] <feats-rxfilename> <feats-wxfilename>
:
:
```

  - **after you complete checkpoint 1-4) successfully, then** execute the main experiment script.

```
./run.sh
```

5. **Explanation about each stage**

- stage -1: data download
  - The downloaded data is stored in `downloads`

```
if [ ${stage} -le -1 ]; then
    echo "stage -1: Data Download"
    mkdir -p ${datadir}
    local/download_and_untar.sh ${datadir} ${data_url}
fi
```

- stage 0: data preparation

```
echo "stage 0: Data preparation"
mkdir -p data/{train,test} exp

if [ ! -f ${an4_root}/README ]; then
    echo Cannot find an4 root! Exiting...
    exit 1
fi

python local/data_prep.py ${an4_root}
${KALDI_ROOT}/tools/sph2pipe_v2.5/sph2pipe

for x in test train; do
    for f in text wav.scp utt2spk; do
```

```
        sort data/${x}/${f} -o data/${x}/${f}
    done
    utils/utt2spk_to_spk2utt.pl data/${x}/utt2spk > data/${x}/spk2utt
done
```

  - The `data` structure is exactly same as the one used in Kaldi, e.g.,

```
$ ls data/train_nodev/
cmvn.ark  feats.scp  spk2utt  text utt2spk  wav.scp
```

- stage 1: feature extraction

```
if [ ${stage} -le 1 ]; then
    ### Task dependent. You have to design training and dev sets by
yourself.
    ### But you can utilize Kaldi recipes in most cases
    echo "stage 1: Feature Generation"
    fbankdir=fbank
    # Generate the fbank features; by default 80-dimensional fbanks with
pitch on each frame
    for x in test train; do
        steps/make_fbank_pitch.sh --cmd "$train_cmd" --nj 8 data/${x}
exp/make_fbank/${x} ${fbankdir}
    done

    # make a dev set
    utils/subset_data_dir.sh --first data/train 100 data/${train_dev}
    n=$[`cat data/train/text | wc -l` - 100]
    utils/subset_data_dir.sh --last data/train ${n} data/${train_set}

    # compute global CMVN
    compute-cmvn-stats scp:data/${train_set}/feats.scp
data/${train_set}/cmvn.ark

    # dump features
    dump.sh --cmd "$train_cmd" --nj 8 --do_delta $do_delta \
        data/${train_set}/feats.scp data/${train_set}/cmvn.ark
exp/dump_feats/train ${feat_tr_dir}
    dump.sh --cmd "$train_cmd" --nj 8 --do_delta $do_delta \
        data/${train_dev}/feats.scp data/${train_set}/cmvn.ark
exp/dump_feats/dev ${feat_dt_dir}
    for rtask in ${recog_set}; do
        feat_recog_dir=${dumpdir}/${rtask}/delta${do_delta}; mkdir -p
${feat_recog_dir}
```

```
        dump.sh --cmd "$train_cmd" --nj 8 --do_delta $do_delta \
            data/${rtask}/feats.scp data/${train_set}/cmvn.ark
exp/dump_feats/recog/${rtask} \
            ${feat_recog_dir}
    done
fi
```

  - use Kaldi feature extraction
- stage 2: dictionary and json data preparation

```
if [ ${stage} -le 2 ]; then
    ### Task dependent. You have to check non-linguistic symbols used in
the corpus.
    echo "stage 2: Dictionary and Json Data Preparation"
    mkdir -p data/lang_1char/
    echo "<unk> 1" > ${dict} # <unk> must be 1, 0 will be used for "blank"
in CTC
    text2token.py -s 1 -n 1 data/${train_set}/text | cut -f 2- -d" " | tr
" " "\n" \
    | sort | uniq | grep -v -e '^\s*$' | awk '{print $0 " " NR+1}' >>
${dict}
    wc -l ${dict}

    # make json labels
    data2json.sh --feat ${feat_tr_dir}/feats.scp \
         data/${train_set} ${dict} > ${feat_tr_dir}/data.json
    data2json.sh --feat ${feat_dt_dir}/feats.scp \
         data/${train_dev} ${dict} > ${feat_dt_dir}/data.json
fi
```

  - create graphme dictionary
  - [ ] **checkpoint 5)**: check the dictionary, which are composed of the alphabet and special symbols (space and unk symbols)

```
$ cat data/lang_1char/train_nodev_units.txt
<unk> 1
<space> 2
A 3
B 4
C 5
D 6
E 7
F 8
G 9
```

```
H 10
I 11
J 12
K 13
L 14
M 15
N 16
O 17
P 18
Q 19
R 20
S 21
T 22
U 23
V 24
W 25
X 26
Y 27
Z 28
```

- create json files, which contain all annotation information (transcriptions, feature paths, speaker IDs, etc.)

- [ ] **checkpoint 6)**: check the json file

```
$ less dump/train_nodev/deltafalse/data.json
{
    "utts": {
        "mtxj-an377-b": { # utterance ID
            "utt2spk": "mtxj", # speaker ID
            "input": [
                {
                    "shape": [
                        368,
                        83
                    ],
                    "feat":
"/export/a08/shinji/201707e2e/espnet_tutorial/egs/an4/asr1/dump/train_node
v/deltafalse/feats.8.ark:1923825",
                    "name": "input1"
                }
            ],
            "output": [
                {
                    "text": "RUBOUT J B X R Z NINE TWENTY",
                    "shape": [
```

```
                        28,
                        30
                    ],
                    "name": "target1",
                    "token": "R U B O U T <space> J <space> B <space> X
<space> R <space> Z <space> N I N E <space> T W E N T Y",
                    "tokenid": "20 23 4 17 23 22 2 12 2 4 2 26 2 20 2 28 2
16 11 16 7 2 22 25 7 16 22 27"
                }
            ]
        },
```

- stage 3: network training

```
if [ ${stage} -le 3 ]; then
    echo "stage 3: Network Training"
    ${cuda_cmd} --gpu ${ngpu} ${expdir}/train.log \
        asr_train.py \
        --ngpu ${ngpu} \
        --backend ${backend} \
        --outdir ${expdir}/results \
        --debugmode ${debugmode} \
        --dict ${dict} \
        --debugdir ${expdir} \
        --minibatches ${N} \
        --verbose ${verbose} \
        --resume ${resume} \
        --train-json ${feat_tr_dir}/data.json \
        --valid-json ${feat_dt_dir}/data.json \
        --etype ${etype} \
        --elayers ${elayers} \
        --eunits ${eunits} \
        --eprojs ${eprojs} \
        --subsample ${subsample} \
        --dlayers ${dlayers} \
        --dunits ${dunits} \
        --atype ${atype} \
        --aconv-chans ${aconv_chans} \
        --aconv-filts ${aconv_filts} \
        --mtlalpha ${mtlalpha} \
        --batch-size ${batchsize} \
        --maxlen-in ${maxlen_in} \
        --maxlen-out ${maxlen_out} \
        --opt ${opt} \
        --epochs ${epochs}
```

```
fi
```

- main training of end-to-end ASR
- several important options
  - `--backend ${backend}`: DNN backend, chainer or pytorch
  - `--elayers ${elayers}`: number of encoder BLSTM layers
  - `--eunits ${eunits}`: number of encoder BLSTM units
  - `--eprojs ${eprojs}`: number of linear layer outputs after every BLSTM layer
  - `--subsample ${subsample}`: subsampling of the encoder BLSTM layers (1_2_2_1_1: we downsample the input of 2nd and 3rd layers with a factor of 2)
  - `--dlayers ${dlayers}`: number of decoder LSTM layers
  - `--dunits ${dunits}`: number of decoder LSTM units
  - `--atype ${atype}`: attention type (location)
  - `--mtlalpha`: tune the CTC weight
  - `--batch-size ${batchsize}`: batch size
  - `--opt ${opt}`: optimizer type

- [ ] **checkpoint 7)**: monitor training. The training takes approximately 15min. per epochs.

```
$ tail -f
exp/train_nodev_blstmp_e4_subsample1_2_2_1_1_unit320_proj320_d1_unit300_lo
cation_aconvc10_aconvf100_mtlalpha0.5_adadelta_bs30_mli800_mlo150/train.lo
g
```

- initial training:

```
2018-06-15 09:32:34,381 (e2e_asr_attctc_th:467) INFO: CTC input lengths:
Variable containing: 62 62 62 62 62 62 62 62 62 62 62 62 62 62 62 62 62 62
62 62 62 62 62 62 62 62 62 62 62 62[torch.IntTensor of
size 30]
2018-06-15 09:32:34,381 (e2e_asr_attctc_th:468) INFO: CTC output lengths:
Variable containing: 25 11  9 12 23 13 35 20 13  9 14 39 21 29  9 40 20  7
11  5 11  9 15 11 17 24 32 31 32 29[torch.IntTensor of
size 30]
2018-06-15 09:32:34,409 (e2e_asr_attctc_th:474) INFO: ctc
loss:163.163162231
2018-06-15 09:32:34,418 (e2e_asr_attctc_th:1674) INFO: Decoder input
lengths:  [62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62,
62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62]
2018-06-15 09:32:34,418 (e2e_asr_attctc_th:1675) INFO: Decoder output
lengths: [26, 12, 10, 13, 24, 14, 36, 21, 14, 10, 15, 40, 22, 30, 10, 41,
21, 8, 12, 6, 12, 10, 16, 12, 18, 25, 33, 32, 33, 30]
```

```
2018-06-15 09:32:34,608 (e2e_asr_attctc_th:1709) INFO: att loss:
66.6240[torch.cuda.FloatTensor of size 1 (GPU 0)]
2018-06-15 09:32:34,609 (e2e_asr_attctc_th:1724) INFO: groundtruth[0]:
RUBOUT<space>N<space>Z<space>X<space>L<space>THIRTY<space>ONE<eos>
2018-06-15 09:32:34,609 (e2e_asr_attctc_th:1725) INFO: prediction [0]:
XNRNRR<eos>JJJJJJJJJ<eos>JJNJJJJQ
2018-06-15 09:32:34,609 (e2e_asr_attctc_th:1724) INFO: groundtruth[1]:
N<space>E<space>W<space>E<space>L<space>L<eos>
2018-06-15 09:32:34,609 (e2e_asr_attctc_th:1725) INFO: prediction [1]:
XMJ<unk>M<blank><blank>B<blank>J<blank>J
2018-06-15 09:32:34,609 (e2e_asr_attctc_th:1724) INFO: groundtruth[2]:
C<space>H<space>R<space>I<space>S<eos>
2018-06-15 09:32:34,610 (e2e_asr_attctc_th:1725) INFO: prediction [2]:
XOMMMNMJMJ
2018-06-15 09:32:34,610 (e2e_asr_attctc_th:1724) INFO: groundtruth[3]:
A<space>A<space>I<space>L<space>ZERO<eos>
2018-06-15 09:32:34,610 (e2e_asr_attctc_th:1725) INFO: prediction [3]:
XKJJJMJJJUNN
2018-06-15 09:32:34,610 (e2e_asr_attctc_th:1724) INFO: groundtruth[4]:
ENTER<space>SEVEN<space>TWO<space>ONE<space>SIX<eos>
2018-06-15 09:32:34,610 (e2e_asr_attctc_th:1725) INFO: prediction [4]:
XXX<eos>XNQPXXXQJ<eos><eos>Q<blank>BQMMMQJ
2018-06-15 09:32:34,611 (e2e_asr_attctc_th:96) INFO: mtl
loss:114.893569946
2018-06-15 09:32:34,759 (asr_pytorch:129) INFO: grad norm=46.892091356
```

- final training:

```
2018-06-15 09:35:16,868 (e2e_asr_attctc_th:467) INFO: CTC input lengths:
Variable containing: 47 45 45 45 45 45 45 45 45 45 45 45 45 45 45 45 45 45
45 45 45 45 45 45 45 42 42 42 42 42[torch.IntTensor of
size 30]
2018-06-15 09:35:16,868 (e2e_asr_attctc_th:468) INFO: CTC output lengths:
Variable containing: 24 21 10 21 20 20 22  9  7 22 22 17 22 22 22 22 20 11
22 22 16 13 15 10 22 15 18 22 22 19[torch.IntTensor of
size 30]
2018-06-15 09:35:16,870 (e2e_asr_attctc_th:474) INFO: ctc
loss:26.512140274
2018-06-15 09:35:16,878 (e2e_asr_attctc_th:1674) INFO: Decoder input
lengths:  [47, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45,
45, 45, 45, 45, 45, 45, 45, 45, 45, 42, 42, 42, 42, 42]
2018-06-15 09:35:16,878 (e2e_asr_attctc_th:1675) INFO: Decoder output
lengths: [25, 22, 11, 22, 21, 21, 23, 10, 8, 23, 23, 18, 23, 23, 23, 23,
21, 12, 23, 23, 17, 14, 16, 11, 23, 16, 19, 23, 23, 20]
```

```
2018-06-15 09:35:16,909 (e2e_asr_attctc_th:1709) INFO: att loss:
13.7800[torch.cuda.FloatTensor of size 1 (GPU 0)]
2018-06-15 09:35:16,910 (e2e_asr_attctc_th:1724) INFO: groundtruth[0]:
ENTER<space>SEVEN<space>THIRTY<space>EIGHT<eos>
2018-06-15 09:35:16,910 (e2e_asr_attctc_th:1725) INFO: prediction [0]:
ENTER<space>SEVEN<space>TWRRTY<space>SIGHT<space>
2018-06-15 09:35:16,910 (e2e_asr_attctc_th:1724) INFO: groundtruth[1]:
THREE<space>FOUR<space>EIGHT<space>ZERO<eos>
2018-06-15 09:35:16,910 (e2e_asr_attctc_th:1725) INFO: prediction [1]:
EWREE<space>TOUR<space>TIGHT<space>TERO<eos>
2018-06-15 09:35:16,910 (e2e_asr_attctc_th:1724) INFO: groundtruth[2]:
ENTER<space>NINE<eos>
2018-06-15 09:35:16,910 (e2e_asr_attctc_th:1725) INFO: prediction [2]:
ONTER<space>OINE<space>
2018-06-15 09:35:16,911 (e2e_asr_attctc_th:1724) INFO: groundtruth[3]:
FIFTY<space>THREE<space>FORTY<space>TWO<eos>
2018-06-15 09:35:16,911 (e2e_asr_attctc_th:1725) INFO: prediction [3]:
EOVTY<space>TWREE<space>TOUTY<space>SWO<space>
2018-06-15 09:35:16,911 (e2e_asr_attctc_th:1724) INFO: groundtruth[4]:
ONE<space>FIVE<space>TWO<space>TWO<space>ONE<eos>
2018-06-15 09:35:16,911 (e2e_asr_attctc_th:1725) INFO: prediction [4]:
ONE<space>FIVE<space>TWO<space>OWO<space>ONE<space>
2018-06-15 09:35:16,912 (e2e_asr_attctc_th:96) INFO: mtl loss:20.146074295
2018-06-15 09:35:17,008 (asr_pytorch:129) INFO: grad norm=28.6609776055
```

- you can see that the prediction result is getting better
- approximately 15min. per epochs. (20 epochs X 15min. = 5 hours. It will not be finished during the lab)

- [ ] **checkpoint 8)**: check costs and accuracies per epoch

```
exp/train_nodev_blstmp_e4_subsample1_2_2_1_1_unit320_proj320_d1_unit300_lo
cation_aconvc10_aconvf100_mtlalpha0.5_adadelta_bs30_mli800_mlo150/results/
log
```

and/or

```
exp/train_nodev_blstmp_e4_subsample1_2_2_1_1_unit320_proj320_d1_unit300_lo
cation_aconvc10_aconvf100_mtlalpha0.5_adadelta_bs30_mli800_mlo150/results/
{acc,loss}.png
```

- [ ] **checkpoint 9)**: check attention weights (**but unfortunately AN4 data is too small to fully train the attention model, and we cannnot find clear attention patterns from this data**)

```
exp/train_nodev_blstmp_e4_subsample1_2_2_1_1_unit320_proj320_d1_unit300_lo
cation_aconvc10_aconvf100_mtlalpha0.5_adadelta_bs30_mli800_mlo150/results/
att_ws/*.png
```

- stage 4: decoding

```
if [ ${stage} -le 4 ]; then
    echo "stage 4: Decoding"
    nj=8

    for rtask in ${recog_set}; do
    (

decode_dir=decode_${rtask}_beam${beam_size}_e${recog_model}_p${penalty}_le
n${minlenratio}-${maxlenratio}_ctcw${ctc_weight}
        feat_recog_dir=${dumpdir}/${rtask}/delta${do_delta}

        # split data
        data=data/${rtask}
        split_data.sh --per-utt ${data} ${nj};
        sdata=${data}/split${nj}utt;

        # make json labels for recognition
        for j in `seq 1 ${nj}`; do
            data2json.sh --feat ${feat_recog_dir}/feats.scp \
                ${sdata}/${j} ${dict} > ${sdata}/${j}/data.json
        done

        #### use CPU for decoding
        ngpu=0

        ${decode_cmd} JOB=1:${nj}
${expdir}/${decode_dir}/log/decode.JOB.log \
            asr_recog.py \
            --ngpu ${ngpu} \
            --backend ${backend} \
            --debugmode ${debugmode} \
            --verbose ${verbose} \
            --recog-json ${sdata}/JOB/data.json \
            --result-label ${expdir}/${decode_dir}/data.JOB.json \
            --model ${expdir}/results/model.${recog_model}  \
            --model-conf ${expdir}/results/model.conf  \
            --beam-size ${beam_size} \
            --penalty ${penalty} \
```

```
            --maxlenratio ${maxlenratio} \
            --minlenratio ${minlenratio} \
            --ctc-weight ${ctc_weight} \
            &
        wait

        score_sclite.sh ${expdir}/${decode_dir} ${dict}

    ) &
    done
    wait
    echo "Finished"
fi
```

- decoding is perfomed with multiple (8) CPUs for development (`test_dev`) and evaluation (`test`) test sets.
- several important options
  - `--beam-size ${beam_size}`: beam size
  - `--ctc-weight ${ctc_weight}`: CTC score weight during beam search

- [ ] **checkpoint 10)**: check final results. This would be appeared in the terminal when we finish all experiments, e.g.,

```
2018-06-20 12:45:07,057 (concatjson:28) INFO: new json has 100 utterances
2018-06-20 12:45:07,425 (json2trn:22) INFO: reading
exp/train_nodev_blstmp_e4_subsample1_2_2_1_1_unit320_proj320_d1_unit300_loc
ation_aconvc10_aconvf100_mtlalpha0.5_adadelta_bs30_mli800_mlo150/decode_tra
in_dev_beam20_eacc.best_p0.0_len0.0-0.0_ctcw0.5/data.json
2018-06-20 12:45:07,427 (json2trn:26) INFO: reading
data/lang_1char/train_nodev_units.txt
2018-06-20 12:45:07,428 (json2trn:34) INFO: writing hyp trn to
exp/train_nodev_blstmp_e4_subsample1_2_2_1_1_unit320_proj320_d1_unit300_loc
ation_aconvc10_aconvf100_mtlalpha0.5_adadelta_bs30_mli800_mlo150/decode_tra
in_dev_beam20_eacc.best_p0.0_len0.0-0.0_ctcw0.5/hyp.trn
2018-06-20 12:45:07,428 (json2trn:35) INFO: writing ref trn to
exp/train_nodev_blstmp_e4_subsample1_2_2_1_1_unit320_proj320_d1_unit300_loc
ation_aconvc10_aconvf100_mtlalpha0.5_adadelta_bs30_mli800_mlo150/decode_tra
in_dev_beam20_eacc.best_p0.0_len0.0-0.0_ctcw0.5/ref.trn
write a CER (or TER) result in
exp/train_nodev_blstmp_e4_subsample1_2_2_1_1_unit320_proj320_d1_unit300_loc
ation_aconvc10_aconvf100_mtlalpha0.5_adadelta_bs30_mli800_mlo150/decode_tra
in_dev_beam20_eacc.best_p0.0_len0.0-0.0_ctcw0.5/result.txt
|         SPKR           |        # Snt                 # Wrd         |
Corr                   Sub                    Del                    Ins
Err                 S.Err  |
```

```
|         Sum/Avg         |         100                    1915          |
77.1                    7.5                    15.4                    0.4
23.3                    79.0  |
2018-06-20 12:45:12,744 (concatjson:28) INFO: new json has 130 utterances
2018-06-20 12:45:13,547 (json2trn:22) INFO: reading
exp/train_nodev_blstmp_e4_subsample1_2_2_1_1_unit320_proj320_d1_unit300_loc
ation_aconvc10_aconvf100_mtlalpha0.5_adadelta_bs30_mli800_mlo150/decode_tes
t_beam20_eacc.best_p0.0_len0.0-0.0_ctcw0.5/data.json
2018-06-20 12:45:13,554 (json2trn:26) INFO: reading
data/lang_1char/train_nodev_units.txt
2018-06-20 12:45:13,555 (json2trn:34) INFO: writing hyp trn to
exp/train_nodev_blstmp_e4_subsample1_2_2_1_1_unit320_proj320_d1_unit300_loc
ation_aconvc10_aconvf100_mtlalpha0.5_adadelta_bs30_mli800_mlo150/decode_tes
t_beam20_eacc.best_p0.0_len0.0-0.0_ctcw0.5/hyp.trn
2018-06-20 12:45:13,555 (json2trn:35) INFO: writing ref trn to
exp/train_nodev_blstmp_e4_subsample1_2_2_1_1_unit320_proj320_d1_unit300_loc
ation_aconvc10_aconvf100_mtlalpha0.5_adadelta_bs30_mli800_mlo150/decode_tes
t_beam20_eacc.best_p0.0_len0.0-0.0_ctcw0.5/ref.trn
write a CER (or TER) result in
exp/train_nodev_blstmp_e4_subsample1_2_2_1_1_unit320_proj320_d1_unit300_loc
ation_aconvc10_aconvf100_mtlalpha0.5_adadelta_bs30_mli800_mlo150/decode_tes
t_beam20_eacc.best_p0.0_len0.0-0.0_ctcw0.5/result.txt
|        SPKR            |        # Snt                # Wrd          |
Corr                    Sub                    Del                    Ins
Err                 S.Err         |
|        Sum/Avg         |         130                  2565          |
84.0                    6.0                  10.0                    1.1
17.0                  68.5          |
```

6. **Modification of configurations**
    - try to optimize some configurations by tuning the following hyperparameters (do not have to try everything)
    - use other features
        - modify Kaldi feature extraction at stage 1
        - run an experiment from stage 1

        ```
        ./run.sh --stage 1
        ```

    - change model topologies
        - Number of encoder layers

        ```
        ./run.sh --stage 3 --elayers XXX
        ```

- Number of encoder units

```
./run.sh --stage 3 --eunits XXX
```

- Number of decoder layers

```
./run.sh --stage 3 --dlayers XXX
```

- Number of decoder units

```
./run.sh --stage 3 --dunits XXX
```

- change attention type (choices=['noatt', 'dot', 'add', 'location', 'coverage', 'coverage_location', 'location2d', 'location_recurrent', 'multi_head_dot', 'multi_head_add', 'multi_head_loc', 'multi_head_multi_res_loc']. Probably some of them would not be working correctly.)

```
./run.sh --stage 3 --atype XXX
```

- optmization (Adadelta -> Adam)

```
./run.sh --stage 3 --opt adam
```

- tune the CTC-attention weights

```
# change CTC attention weights
$ ./run.sh --stage 3 --mtlalpha XXX --ctc_weight XXX

# CTC mode
$ ./run.sh --stage 3 --mtlalpha 1.0 --ctc_weight 1.0 --recog_model
loss.best

# attention mode
$ ./run.sh --stage 3 --mtlalpha 0.0 --ctc_weight 0.0
```

- [ ] **checkpoint 11)**: report your best number (CER) for the test set to me shinjiw@ieee.org

7. **Run the AN4 recipe with GPUs (optional)**

- ESPnet is designed to be used with the GPU. If you have enough knowledge to use GPUs in the CLSP cluster, **ask instructors in advance**, then perform GPU experiments.

```
./run.sh --stage 3 --ngpu 1
```