Instructions

1 Installing AST system

1.1 Install conda environment

To create a `conda` environment with Python 3, run the following command:

```
conda create --name ast python=3
```

If other location than the default one is preferred for the conda environment, the option `--prefix PATH` can be added. For example:

```
conda create --prefix \home\virtualenv\ast python=3
```

1.2 Install dependencies in the conda environment

Note that the following commands need to be run from a GPU server. Next, activate the new environment and install the required dependencies as follows:

```
source activate ast
pip install cupy-cuda91
pip install chainer
```

Since the AST system will be trained on GPUs, we need to make sure that `chainer` allows using GPUs. To check this, we enter the `python` interpreter (by just typing `python` in the command line and pressing `Enter`):

```
Python 3.6.2
Type "help", "copyright", "credits" or "license" for more information.
>>> import chainer
>>> chainer.backends.cuda.available
```

```
True
>>> chainer.backends.cuda.cudnn_enabled
True
>>>
```

Finally, install `NLTK` and `tqdm`. `NLTK` is used to extract stop word lists for target languages and to compute evaluation metrics such as BLEU score. The package `tqdm` provides a better visualisation of the training progress by displaying and updating a progress bar in the terminal.

```
    conda install nltk
    conda install tqdm
```

2 Creating speech features for pretraining the AST system

2.1 Preparing MFCC files from GP

In this section we describe the steps needed to obtain MFCC Kaldi archives and to convert them into Numpy archives, which is the required format for the input to the AST system.

2.1.1 Creating MFCCs

First, 13-dimensional MFCCs are obtained by using the script `steps/make_mfcc.sh`. Before running the script, modify the file `conf/mfcc.conf` to contain:

```
--use-energy=false   # only non-default option.
--
```

This script will create a script-file [1] (with extension `scp`), named `data/train/feats.scp`, which is the concatenation of the `scp` files from the mfcc directory, i.e. from Kaldi's `steps/make_mfcc.sh`:

```
for n in $(seq $nj); do
  cat $mfccdir/raw_mfcc_$name.$n.scp || exit 1;
done > $data/feats.scp || exit 1
```

2.1.2 Applying CMVN

The next step is to apply Cepstral Mean Variance Normalisation (CMVN) on top of the previously-obtained MFCC features:

```
apply-cmvn --norm-vars=true --utt2spk=ark:data/train/utt2spk \
    scp:data/train/cmvn.scp scp:data/train/feats.scp \
    ark:- | copy-feats ark:- \
    ark,t:all_train_cmvn.ark
```

Note: for this part use script `/safe-copy-ast/gp_data_cmvn/apply_cmvn.sh`, which also works for multiple languages.

[1]Full description here: `http://kaldi-asr.org/doc/io.html#io_sec_scp`.

2.1.3 Checking Kaldi ark and scp dimensions

To check that the ark and scp files are in the correct format, we can run the following:

```
feat-to-dim scp:train/feats.scp -
feat-to-dim ark:all_train_cmvn.ark -
```

Note: `train/feats.scp` and `all_train_cmvn.ark` should be 13-dimensional, while `train/cmvn.scp` should have 14 dimensions.

2.1.4 Converting Kaldi archives to Numpy archives

For each dataset (train, dev, and test), we need to execute the following command, using the previously-obtained CMVN MFCC features:

```
nice python ./kaldi_io.py ./ast/data/gpfr/all_train_cmvn.ark io_arks/gpfr/train
```

Note that this command will produce `np` files stored in a separate directory associated with each dataset.

2.1.5 Processing the date text and mapping from text to utterance IDs

We move the `np` files in a folder called `pretrain_AST_input`. For each GlobalPhone language ¡lang¿, we store a directory with the following name `gp<lang>`.

In the following example, we will be working with the French corpus from GlobalPhone.

First, copy the data `text` files into the directories containing the `np` files.

```
cp ../data/FR/train/text gpFR/train # do this also for dev, test
```

For executing the above command and the steps from previous section, use script `./apply_kaldi_io.sh`.

Then, modify the `languages` fields of `get_clean-text_ids.sh` to `"FR"` and simply run the script afterwards.

2.1.6 Learning BPE on train set

Note that BPE stands for byte pair encoding.

First, clone and install the following repository: `https://github.com/rsennrich/subword-nmt`. After installing it, the command `subword-nmt` should have become available.

Then, learn BPE on train set and apply it on train, dev, test sets:

```
cd ../subword-nmt
./get_bpe.sh
```

2.1.7 Creating dictionaries

```
cd ../ast/preprocessing
python preprocess_gp.py \
   "../../pretrain_AST_input/gpFR/" \
   "../../pretrain_AST_input/gpFR/"
```

Resulting files:

- `info.dict`
- `data.dict`
- `bpe_map.dict`
- `bpe_train_vocab.dict`

2.1.8 Creating `refs`

The text and id references for dev and test sets are created by running the following:

```
cd ../../pretrain_AST_input/gpFR/
mkdir -p refs/dev
cp dev.clean.text refs/dev/ref.en0
cp dev.ids refs/dev/eval.ids

# run all of the above on test set as well
```

```
mkdir -p refs/test
cp test.clean.text refs/test/ref.en0
cp test.ids refs/test/eval.ids
```

3 Using the AST system

Given parallel data between source and target languages and input speech features (e.g. MFCCs), we can train the AST system.

Note that all of the instructions below should be run from the conda environment from Section 1.2.

3.1 Train the AST system

First run the following command:

```
./run_exp.bat experiments/es_en_20h 40
```

The first argument `experiments/es_en_20h` represents the directory which must contain:

- `model_cfg.json`: specifies the hyperparamters of the model: number of hidden layers, number of hidden units, dropout rates, etc.
- `train_cfg.json`: specifies parameters for the training script such as: the input speech features directory, number of training epochs until saving checkpoint, learning rate, added noise levels, GPU id on which the job should be run, etc.

The second parameter specifies for how many epochs the model should be trained; in this case: 40.

Note that for running on afs, using a screen, the following command should be used instead:

```
longjob -28day -nobackground -c \
    "./run_exp.bat experiments/es_en_20h 40"
```

3.2 Translate source into target language, using the trained AST model

Simply execute the following:

```
python beam.py \
    -m experiments/es_en_20h \
    -n 5 \
    -k 5 \
    -w 0.6 \
    -s fisher_dev
```

where:

- -m `dir`: specifies the location of the trained AST model containing the model at best epoch
- -k `integer`: specifies the beam width
- -w `float`: specifies the length normalization weight (depending on it, the shorter sentences will be more or less encouraged)
- -s `dir`: specifies the location of the data for which we want to obtain a translation (e.g. the dev set)

3.3 Evaluate the translation with BLUE score

First, ensure that *mosesdecoder* exists in the current directory. Otherwise, obtain it by running:

```
git clone https://github.com/moses-smt/mosesdecoder.git
```

Then execute the following:

```
mosesdecoder/scripts/generic/multi-bleu.perl \
    data/fisher/refs/fisher_dev/ref.en* < \
    experiments/sp_20hrs_bnf-cr/fisher_dev_beam_N-5_K-5_W-0.60.en
```

where:

- `data/fisher/refs/fisher_dev/ref.en*`: represents the reference translation file(s); recall that one advantage of BLUE score is that it can compare the translation against **multiple** refernce files

- `experiments/sp_20hrs_bnf-cr/fisher_dev_beam_N-5_K-5_W-0.60.en`: represents the model's prediction (i.e. translation)

4 AST system's hyperparameters

In this section, we describe the hyperparameters of the AST system.

- `add_noise`: if it is set to a value larger than 0, then noise is added to the input: a Chainer normal random variable (RV) is created using `np.random.normal` to set the mean to 1 and the standard deviation to the value specified by `add_noise` (so, the `add_noise` parameter represents the stddev of the normal RV); then the input `X` is multiplied by the newly created Chainer normal RV.

- `teach_ratio`: if a randomly generated number is less than the value of `teach_ratio`, then the decoder input is set to be the word observed in the previous time step; otherwise, the decoder input is set to be the predicted word.

- `random_out`: if it is larger than 0, then before computing the loss function value, for each target word the following instructions are executed: if a randomly generated number is greater than the value of `random_out`, then the current target word is replaced by a randomly selected word from the available vocabulary.

- `dropout`: is the value of the argument `p` of the method `dropout`, from the package `chainer.functions`, which "drops input elements randomly with probability p and scales the remaining elements by factor $\frac{1}{1-p}$".[2]. Note: it can be applied, with different values, at the embedding, RNN and output layers; however, it is usually set to 0 at the latter.

[2]Full description here: `https://docs.chainer.org/en/stable/reference/generated/chainer.functions.dropout.html`