SETTING UP KALDI WITH TENSORFLOW FOR SPEAKER VERIFICATION (version 1)

Iván López-Espejo (ivl@es.aau.dk)

This document will guide you through setting up Kaldi with TensorFlow for speaker verification based on x-vectors[1]. This approach consists of a time-delay neural network (TDNN) extracting speaker embeddings (from speech features) that are further scored by means of a probabilistic linear discriminant analysis (PLDA) back-end. While the whole pipeline is implemented in Kaldi, we will replace the Kaldi TDNN-based extractor by another one coded in TensorFlow, which will allow us to more easily integrate our developments on the front-end side.

Kaldi installation

1. Download Kaldi from https://kaldi-asr.org/doc/install.html.
2. Download the Intel Math Kernel Libraries (MKL) for linear algebra operations from https://software.intel.com/content/www/us/en/develop/tools/math-kernel-library/choose-download/linux.html. Extract the file contents and launch the installer by executing `install.sh`. Indicate the path in which MKL will be installed ensuring that you have writing permissions, e.g., `/home/<user>/intel`, and complete the installation.
3. Go to `kaldi/tools/extras` and edit the file `check_dependencies.sh`. In this file, replace the line `MKL_ROOT="${MKL_ROOT:-/opt/intel/mkl}"` by `MKL_ROOT="${MKL_ROOT:-/home/<user>/intel/mkl}"`. Then, execute `cd ..` and `extras/check_dependencies.sh`. Satisfy any possible missing dependencies before continuing.
4. Run `make -j <num_cpu>`.
5. Go to `kaldi/src` and execute `./configure --shared --mkl-root=/home/<user>/intel/mkl`. Be sure that your CUDA version is 9.0 or greater!
6. Finally, install Kaldi by running `make depend -j <num_cpu>` and, then, `make -j <num_cpu>`.

Running a purely Kaldi-based speaker verification system

Kaldi includes a few examples of x-vector-based speaker verification systems. We will run the one using the VoxCeleb 1[2] and 2[3] datasets.

1. Go to http://www.robots.ox.ac.uk/~vgg/data/voxceleb/ and download VoxCeleb 1 and VoxCeleb 2 as explained there. Extract the contents of VoxCeleb 1 and 2 in, e.g., `/home/<user>/voxceleb1` and `/home/<user>/voxceleb2`, respectively. Be sure that you have the following folder structure for VoxCeleb 1:

```
/home/<user>/voxceleb1/
                       |__dev/wav
                       |__test/wav
```

And for VoxCeleb 2:

```
/home/<user>/voxceleb2/
                       |__audio/dev/aac
                       |__audio/test/aac
```

[1] D. Snyder *et al.*, "X-Vectors: Robust DNN Embeddings for Speaker Recognition". In Proc. of ICASSP 2018, https://maelfabien.github.io/assets/litterature/representation/x_vector.pdf

[2] A. Nagrani *et al.*, "VoxCeleb: A Large-Scale Speaker Identification Dataset". In Proc. of Interspeech 2017, http://www.robots.ox.ac.uk/~vgg/publications/2017/Nagrani17/nagrani17.pdf

[3] J. S. Chung *et al.*, "VoxCeleb2: Deep Speaker Recognition". In Proc. of Interspeech 2018, http://www.robots.ox.ac.uk/~vgg/publications/2018/Chung18a/chung18a.pdf

2. Go to https://openslr.org/17/ and download the MUSAN dataset, which will be used for data augmentation purposes. Extract the contents of MUSAN in, e.g., `/home/<user>/musan`.
3. Go to `kaldi/egs/voxceleb/v2`, where the scripts to train an x-vector-based speaker verification system using the VoxCeleb 1 and 2 datasets are. Edit the file `run.sh`. At the beginning of this script, set `voxceleb1_root=/home/<user>/voxceleb1`, `voxceleb2_root=/home/<user>/voxceleb2/audio` and `musan_root=/home/<user>/musan`.
4. In the same folder, edit the file `cmd.sh`. If you are not using a queue system to run your experiments, be sure that you set in this file `export train_cmd="run.pl"`.
5. You will want to exploit your GPUs to train the TDNN-based extractor. To avoid memory allocation issues, first, check the compute mode of your GPUs. For this, execute `nvidia-smi --query | grep 'Compute Mode'`. The most likely is that the GPU compute mode is *default*. If you do not get the output `Compute Mode : Exclusive_Process` for your GPUs, run `nvidia-smi -c 3` to solve it (you will need root permissions for this!). In addition, go to `kaldi/egs/voxceleb/v2/local/nnet3/xvector` and edit `run_xvector.sh`. At the beginning of this script set `use_gpu=true` and, at the end of the script, set the parameter `--use-gpu=wait` for *train_raw_dnn.py*.
6. Finally, go back to `kaldi/egs/voxceleb/v2` and run `./run.sh` to train and test the speaker verification system.

Integrating TensorFlow in the Kaldi pipeline

Finally, we will integrate a TensorFlow TDNN-based extractor in the Kaldi pipeline. That was implemented by researchers from Brno University of Technology[4].

1. Go to `kaldi/egs/voxceleb/v2` and clone the TensorFlow-based speaker embedding extractor repository running `git clone https://github.com/hsn-zeinali/x-vector-kaldi-tf.git`. Then, run `cp -r x-vector-kaldi-tf/local/tf/ local/`.
2. Again, edit `run.sh` and replace `local/nnet3/xvector/run_xvector.sh` by `local/tf/run_xvector.sh`, and (twice) `sid/nnet3/xvector/extract_xvectors.sh` by `local/tf/extract_xvectors.sh`.
3. Go to `kaldi/egs/voxceleb/v2/local/tf` and edit `run_xvector.sh`. Once again, at the beginning of this script set `use_gpu=true` and, at the end of the script, set the parameter `--use-gpu=yes` for *train_dnn.py*. In addition, in Line 50, replace the stage number 4 by 6. Similarly, in Line 87, replace the stage number 6 by 8.
4. To save some time having to fix some version compatibility issues, be sure that you use Python v2, TensorFlow v1 and NumPy 1.16.1.
5. Finally, go back to `kaldi/egs/voxceleb/v2` and run `./run.sh` to train and test the speaker verification system. You should get similar results to the ones obtained using a purely Kaldi-based system.

[4] H. Zeinali *et al.*, "How to Improve your Speaker Embeddings Extractor in Generic Toolkits". In Proc. of ICASSP 2019, https://arxiv.org/pdf/1811.02066.pdf