JSALT19 Speaker Detection Tutorial

Jesús Villalba, Paola García, Phani Sankar Nidadavolu, Saurabh Kataria

06/20/19

Introduction

- Clone:
  - git clone https://github.com/jsalt2019-diadet/jsalt2019-tutorial.git
  - Dependencies:
    - Kaldi: feature and embedding extraction
    - Python3.5
    - Hyperion toolkit: python package with utilities for speaker recognition
  - Create links to depencies in AWS
    - cd jsalt2019-tutorial
    - make_awk_links.sh
      - It will create links to already compiled anaconda3 and Kaldi in the grid.

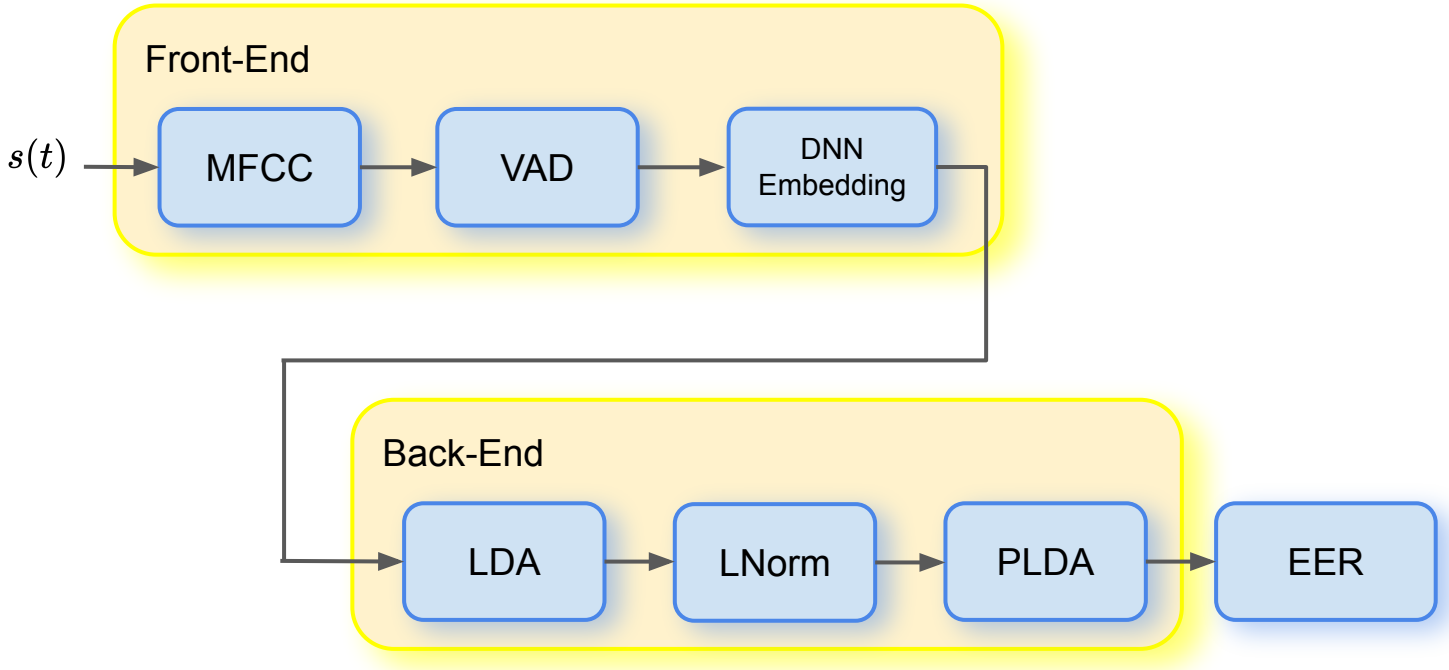
Directory structure

- Egs: recipes
  - sitw_tutorial/v1
    - Speaker verification example.
    - Based on the Speakers in the Wild dataset.
  - callhome_diarization/v1
    - Speaker diarization example.
    - Based on the calhome dataset
- Tools: links to dependencies
  - Hyperion: python tools
    - LDA/PLDA back-end
    - Calibration
  - Kaldi
  - Anaconda Python

```
./jsalt2019-tutorial
./jsalt2019-tutorial/tools
./jsalt2019-tutorial/tools/anaconda
./jsalt2019-tutorial/tools/anaconda/anaconda3.5
./jsalt2019-tutorial/tools/kaldi
./jsalt2019-tutorial/tools/kaldi/kaldi
./jsalt2019-tutorial/tools/hyperion
./jsalt2019-tutorial/tools/hyperion/hyperion
./jsalt2019-tutorial/egs
./jsalt2019-tutorial/egs/sitw_tutorial
./jsalt2019-tutorial/egs/sitw_tutorial/v1
./jsalt2019-tutorial/egs/callhome_diarization
./jsalt2019-tutorial/egs/callhome_diarization/v1
```

SITW Speaker Verification Pipeline

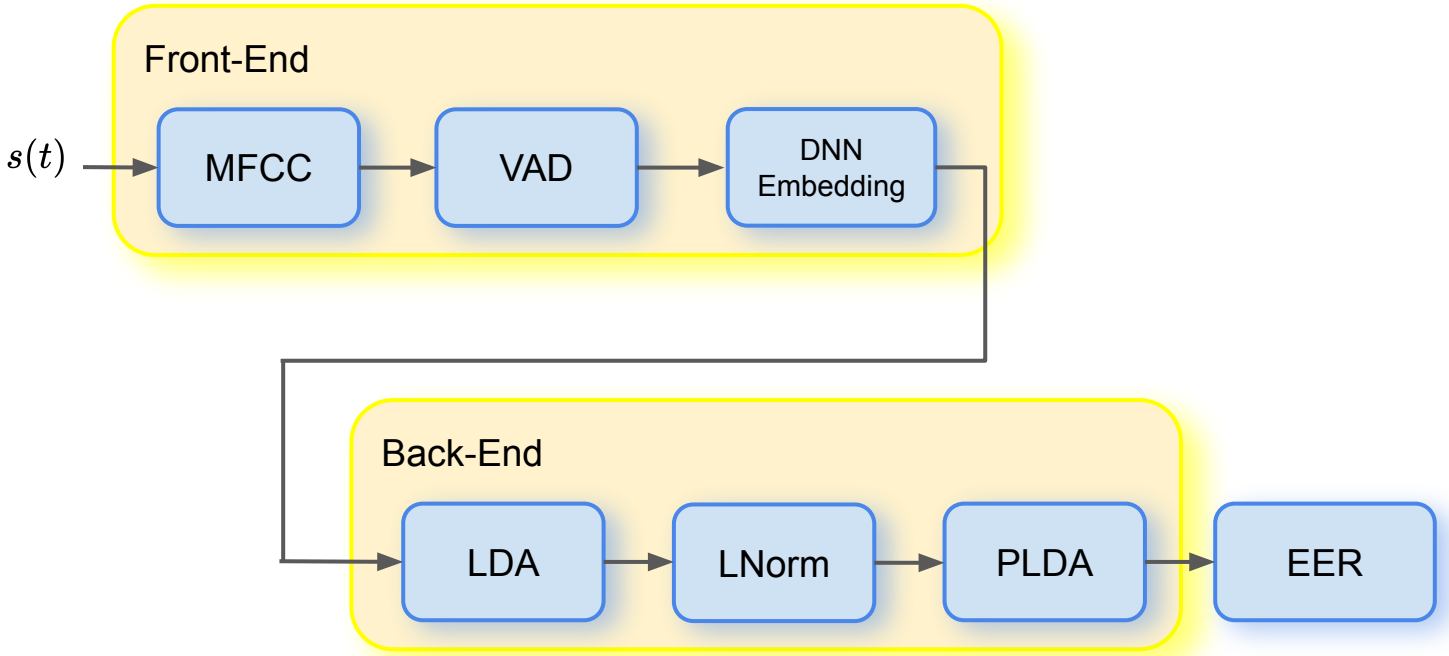
- egs/sitw_tutorial/v1



SITW Speaker Verification Pipeline

- Kaldi style recipe with multiple stages
  - Each run file has a number indicating their position in the pipeline.
  - Each run file has also multiple internal stages

```
run_001_prepare_data.sh
run_002_compute_mfcc_evad.sh
run_010_prepare_xvec_train_data.sh
run_011_train_xvector.sh
run_030_extract_xvectors.sh
run_040_backend_wo_diar.sh
```



SITW: data preparation

- Step 001: run_001_prepare_data.sh
  - Prepares data in kaldi format
  - VoxCeleb:
    - Training data for x-vector and PLDA back-end
    - data/voxceleb_small
      - Wav.scp, utt2spk, spk2utt

```
wav.scp: utterance-id wave-file-path
A.J._Buckley-1zcIwhmdeo4-0000001 /export/corpora/VoxCeleb1/voxceleb1_wav/A.J._Buckley/1zcIwhmdeo4_0000001.wav
A.J._Buckley-9mQ11vBs1wc-0000003 /export/corpora/VoxCeleb1/voxceleb1_wav/A.J._Buckley/9mQ11vBs1wc_0000003.wav
A.J._Buckley-J9lHsKG98U8-0000015 /export/corpora/VoxCeleb1/voxceleb1_wav/A.J._Buckley/J9lHsKG98U8_0000015.wav
A.J._Buckley-Y8hIVOBuels-0000008 /export/corpora/VoxCeleb1/voxceleb1_wav/A.J._Buckley/Y8hIVOBuels_0000008.wav
A.R._Rahman-0_laIeN-Q44-0000001 /export/corpora/VoxCeleb1/voxceleb1_wav/A.R._Rahman/0_laIeN-Q44_0000001.wav
A.R._Rahman-QanuGhOhb9A-0000016 /export/corpora/VoxCeleb1/voxceleb1_wav/A.R._Rahman/QanuGhOhb9A_0000016.wav
A.R._Rahman-RLKKsYiCMvc-0000017 /export/corpora/VoxCeleb1/voxceleb1_wav/A.R._Rahman/RLKKsYiCMvc_0000017.wav
A.R._Rahman-VMaXdHLz5Bk-0000002 /export/corpora/VoxCeleb1/voxceleb1_wav/A.R._Rahman/VMaXdHLz5Bk_0000002.wav
Aamir_Khan-5ablueV_1tw-0000001 /export/corpora/VoxCeleb1/voxceleb1_wav/Aamir_Khan/5ablueV_1tw_0000001.wav
Aamir_Khan-BQxxhq4539A-0000017 /export/corpora/VoxCeleb1/voxceleb1_wav/Aamir_Khan/BQxxhq4539A_0000017.wav
Aamir_Khan-E_6MjfYr0sQ-0000031 /export/corpora/VoxCeleb1/voxceleb1_wav/Aamir_Khan/E_6MjfYr0sQ_0000031.wav
Aamir_Khan-bDxy7bnj_bc-0000007 /export/corpora/VoxCeleb1/voxceleb1_wav/Aamir_Khan/bDxy7bnj_bc_0000007.wav
```

SITW: data preparation

- Step 001: run_001_prepare_data.sh
  - VoxCeleb:
    - Training data for x-vector and PLDA back-end
    - Wav.scp, utt2spk, spk2utt
      - Sorted by speaker and utterance id

```
utt2spk: utterance-id spk-id
A.J._Buckley-1zcIwhmdeo4-0000001 A.J._Buckley
A.J._Buckley-9mQ11vBs1wc-0000003 A.J._Buckley
A.J._Buckley-J9lHsKG98U8-0000015 A.J._Buckley
A.J._Buckley-Y8hIVOBuels-0000008 A.J._Buckley
A.R._Rahman-0_laIeN-Q44-0000001 A.R._Rahman
A.R._Rahman-QanuGhOhb9A-0000016 A.R._Rahman
A.R._Rahman-RLKKsYiCMvc-0000017 A.R._Rahman
A.R._Rahman-VMaXdHLz5Bk-0000002 A.R._Rahman
Aamir_Khan-5ablueV_1tw-0000001 Aamir_Khan
Aamir_Khan-BQxxhq4539A-0000017 Aamir_Khan
Aamir_Khan-E_6MjfYr0sQ-0000031 Aamir_Khan
Aamir_Khan-bDxy7bnj_bc-0000007 Aamir_Khan
Aaron_Tveit-6WxS8rpNjmk-0000001 Aaron_Tveit
Aaron_Tveit-JKMfqmjUwMU-0000007 Aaron_Tveit
Aaron_Tveit-lu_eVSfv3Tg-0000001 Aaron_Tveit
```

```
spk2utt: spk-id utt-ids
A.J._Buckley A.J._Buckley-1zcIwhmdeo4-0000001 A.J._Buckley-9mQ11vBs1wc-0000003 A.J._Buckley-J9lHsKG98U8-0000015
A.J._Buckley-Y8hIVOBuels-0000
008
A.R._Rahman A.R._Rahman-0_laIeN-Q44-0000001 A.R._Rahman-QanuGhOhb9A-0000016 A.R._Rahman-RLKKsYiCMvc-0000017
A.R._Rahman-VMaXdHLz5Bk-0000002
Aamir_Khan Aamir_Khan-5ablueV_1tw-0000001 Aamir_Khan-BQxxhq4539A-0000017 Aamir_Khan-E_6MjfYr0sQ-0000031 Aamir_Khan-
bDxy7bnj_bc-0000007
```

SITW: data preparation

- Step 001: run_001_prepare_data.sh
  - Speakers in the Wild enrollment data:
    - Data used to register speakers into the speaker recognition system
    - data/sitw_dev_enroll
      - wav.scp, utt2spk, spk2utt

```
utt2spk: utt-id spk-id
00033_aynag 00033
00051_wgnio 00051
00235_uhwjv 00235
00264_qofba 00264
00417_vjkji 00417
00489_aqski 00489
00611_ajbav 00611
00650_wpgfo 00650
```

```
wav.scp: utterance-id wave-file-path
00033_aynag /export/corpora/SRI/sitw/dev/audio-wav-16KHz/aynag.wav
00051_wgnio /export/corpora/SRI/sitw/dev/audio-wav-16KHz/wgnio.wav
00235_uhwjv /export/corpora/SRI/sitw/dev/audio-wav-16KHz/uhwjv.wav
00264_qofba /export/corpora/SRI/sitw/dev/audio-wav-16KHz/qofba.wav
00417_vjkji /export/corpora/SRI/sitw/dev/audio-wav-16KHz/vjkji.wav
00489_aqski /export/corpora/SRI/sitw/dev/audio-wav-16KHz/aqski.wav
00611_ajbav /export/corpora/SRI/sitw/dev/audio-wav-16KHz/ajbav.wav
```

```
spk2utt: spk-id utt-ids
00033 00033_aynag
00051 00051_wgnio
00235 00235_uhwjv
00264 00264_qofba
00417 00417_vjkji
00489 00489_aqski
00611 00611_ajbav
00650 00650_wpgfo
01000 01000_uzmkj
01067 01067_vvhdn
01138 01138_wzzqi
```

SITW: data preparation

- Step 001: run_001_prepare_data.sh
  - Speakers in the Wild test data:
    - We get a new recording and we have to decide if the person speaking in that recording is one of the enrolled.
    - data/sitw_dev_test
      - wav.scp, utt2spk, spk2utt
      - In test phase we assume that we don't know the speaker so we use spk-id=utterance-id
      - trials/core-core.lst

```
utt2spk: utt-id spk-id
aahtm aahtm
aaoao aaoao
abvwc abvwc
adfcc adfcc
aeglv aeglv
afbvr afbvr
afecb afecb
afpja afpja
afxbe afxbe
afzwk afzwk
```

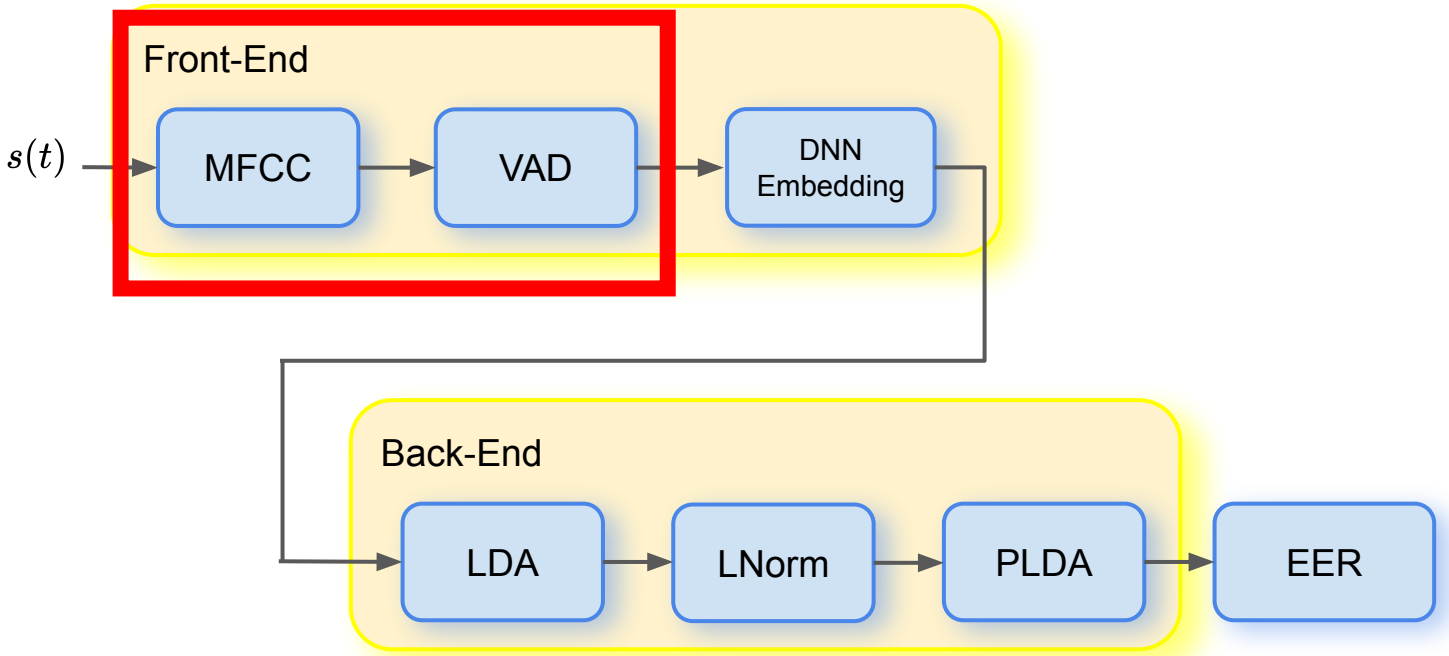
```
spk2utt: spk-id utt-ids
```

```
Trials/core-core.lst:
Enroll-spk test-utt-id target/nontarget
12013 lpnns target
12013 lnmys target
12013 esrdw target
12013 zfykm target
12013 ifisa nontarget
12013 nbzuz nontarget
12013 dkxvg nontarget
12013 qucwb nontarget
```

```
wav.scp: utterance-id wave-fi
aahtm /export/corpora/SRI/sitw/dev/audio-wav-16KHz/aahtm.
aaoao /export/corpora/SRI/sitw/dev/audio-wav-16KHz/aaoao.
abvwc /export/corpora/SRI/sitw/dev/audio-wav-16KHz/abvwc.
```

SITW: compute MFCC features and VAD

- Step 002: run_002_compute_mfcc_evad.sh
  - Compute MFCC features
  - Energy VAD



SITW: compute MFCC features and VAD

- Step 002: run_002_compute_mfcc_evad.sh
  - Compute MFCC features
  - Energy VAD
    - Threshold over the energy based on the mean energy of the full utterance.

```
for name in voxceleb_small
do
    steps/make_mfcc.sh --write-utt2num-frames true --mfcc-config conf/mfcc_16k.conf \
                       --nj 40 --cmd "$train_cmd" \
                      data/${name} exp/make_mfcc $mfccdir
    utils/fix_data_dir.sh data/${name}
    steps_fe/compute_vad_decision.sh --nj 30 --cmd "$train_cmd" \
                                     data/${name} exp/make_vad $vaddir
    utils/fix_data_dir.sh data/${name}
done
```

SITW: compute MFCC features and VAD

- Step 002: run_002_compute_mfcc_evad.sh
  - Adds 2 new files to each data directory
    - feat.scp, vad.scp
    - List files that allow to locate the MFCC/VAD matrix in Kaldi Ark files (binary files)

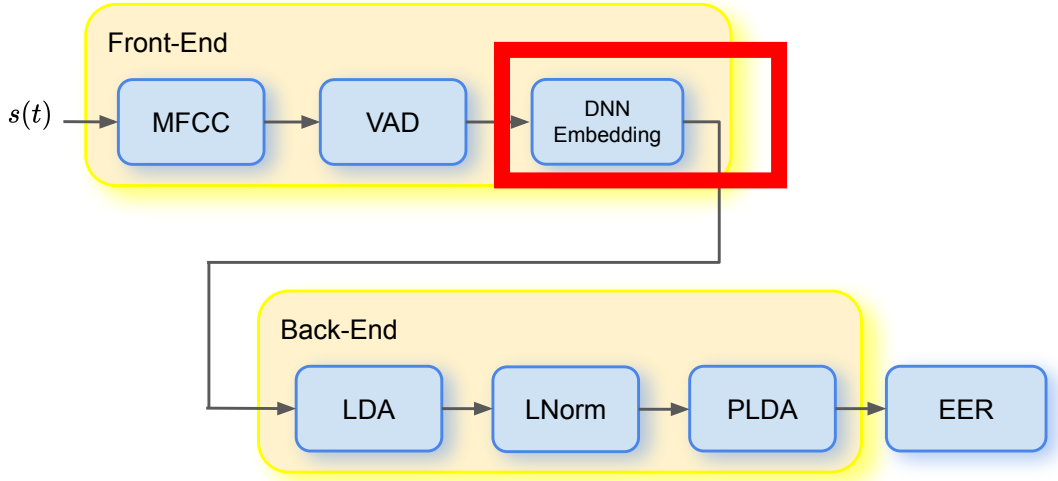
```
feats.scp:
Utterance-id ark-file-where-feature-matrix-is-located:byte-where-feature-is-located

A.J._Buckley-1zcIwhmdeo4-0000001 /export/b15/janto/jsalt2019-tutorial/egs/sitw_tutorial/v1/mfcc/raw_mfcc_voxceleb_small.1.ark:33
A.J._Buckley-9mQ11vBs1wc-0000003 /export/b15/janto/jsalt2019-tutorial/egs/sitw_tutorial/v1/mfcc/raw_mfcc_voxceleb_small.1.ark:32887
A.J._Buckley-J9lHsKG98U8-0000015 /export/b15/janto/jsalt2019-tutorial/egs/sitw_tutorial/v1/mfcc/raw_mfcc_voxceleb_small.1.ark:54221
A.J._Buckley-Y8hIVOBuels-0000008 /export/b15/janto/jsalt2019-tutorial/egs/sitw_tutorial/v1/mfcc/raw_mfcc_voxceleb_small.1.ark:79875
A.R._Rahman-0_laIeN-Q44-0000001 /export/b15/janto/jsalt2019-tutorial/egs/sitw_tutorial/v1/mfcc/raw_mfcc_voxceleb_small.1.ark:100088
A.R._Rahman-QanuGhOhb9A-0000016 /export/b15/janto/jsalt2019-tutorial/egs/sitw_tutorial/v1/mfcc/raw_mfcc_voxceleb_small.1.ark:117101
```

SITW: prepares features to train x-vector NNet

- Step 010: run_010_prepare_xvec_train_data.sh
  - Normalize training features with short-time CMN
  - Remove silence frames
  - It creates a new data directory in data/voxceleb_small_no_sil
  - It removes utterances with less than 4 seconds
    - We will train the x-vector with speech chunks between 2-4 seconds.
  - We remove speakers with less than 4 recordings.
    - We need speakers with multiple recordings to learn how speakers voice change from one recording to another.

SITW: train x-vector network

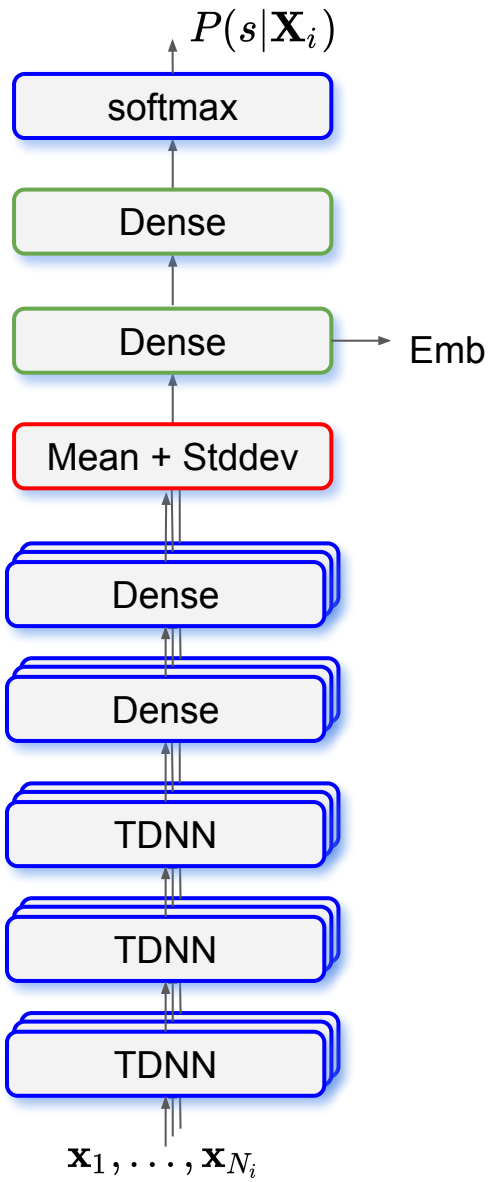
- Step 011: run_011_train_xvector.sh
  - More data preparation
    - Split training recordings into 2-4 secs chunks
    - Select enough random chunks to cover most of the data
    - Chunks uniformly distributed over speakers
    - Shuffle chunks so we present training examples in random order to the network



SITW: train x-vector network

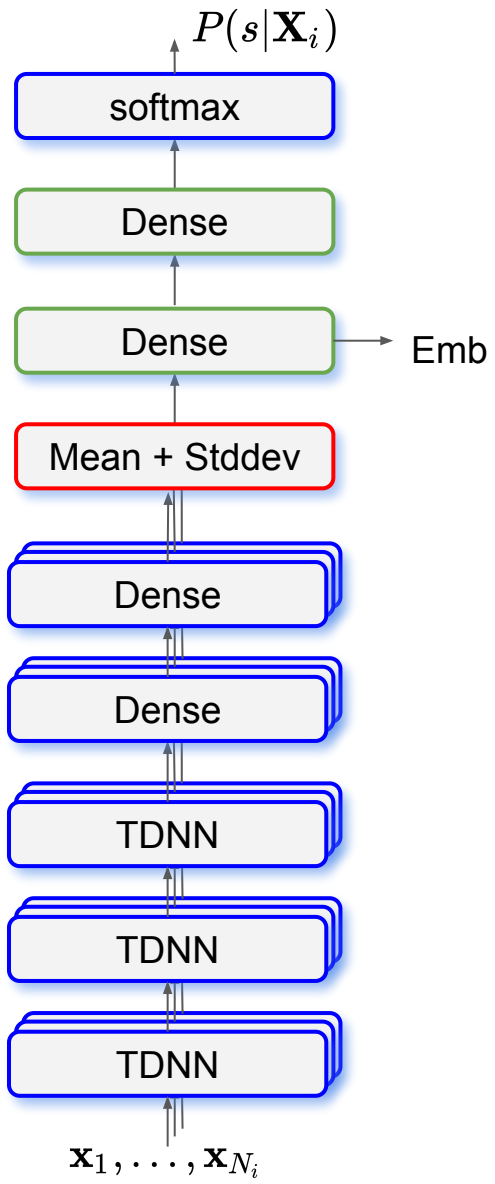
- Step 011: run_011_train_xvector.sh
  - Creates x-vector neural network
  - Network configuration code in
    - steps_kaldi_xvec/run_xvector_2a.1.sh

```
input dim=${feat_dim} name=input
relu-batchnorm-layer name=tdnn1 input=Append(-2,-1,0,1,2) dim=512
relu-batchnorm-layer name=tdnn2 dim=512
relu-batchnorm-layer name=tdnn3 input=Append(-2,0,2) dim=512
relu-batchnorm-layer name=tdnn4 dim=512
relu-batchnorm-layer name=tdnn5 input=Append(-3,0,3) dim=512
relu-batchnorm-layer name=tdnn6 dim=512
relu-batchnorm-layer name=tdnn7 input=Append(-4,0,4) dim=512
relu-batchnorm-layer name=tdnn8 dim=512
relu-batchnorm-layer name=tdnn9 dim=512
relu-batchnorm-layer name=tdnn10 dim=1500

# The stats pooling layer. Layers after this are segment-level.
# In the config below, the first and last argument (0, and ${max_chunk_size})
# means that we pool over an input segment starting at frame 0
# and ending at frame ${max_chunk_size} or earlier.  The other arguments (1:1)
# mean that no subsampling is performed.
stats-layer name=stats config=mean+stddev(0:1:1:${max_chunk_size})

# This is where we usually extract the embedding (aka xvector) from.
relu-batchnorm-layer name=tdnn11 dim=512 input=stats


# This is where another layer the embedding could be extracted
relu-batchnorm-layer name=tdnn12 dim=512
output-layer name=output include-log-softmax=true dim=${num_targets}
```



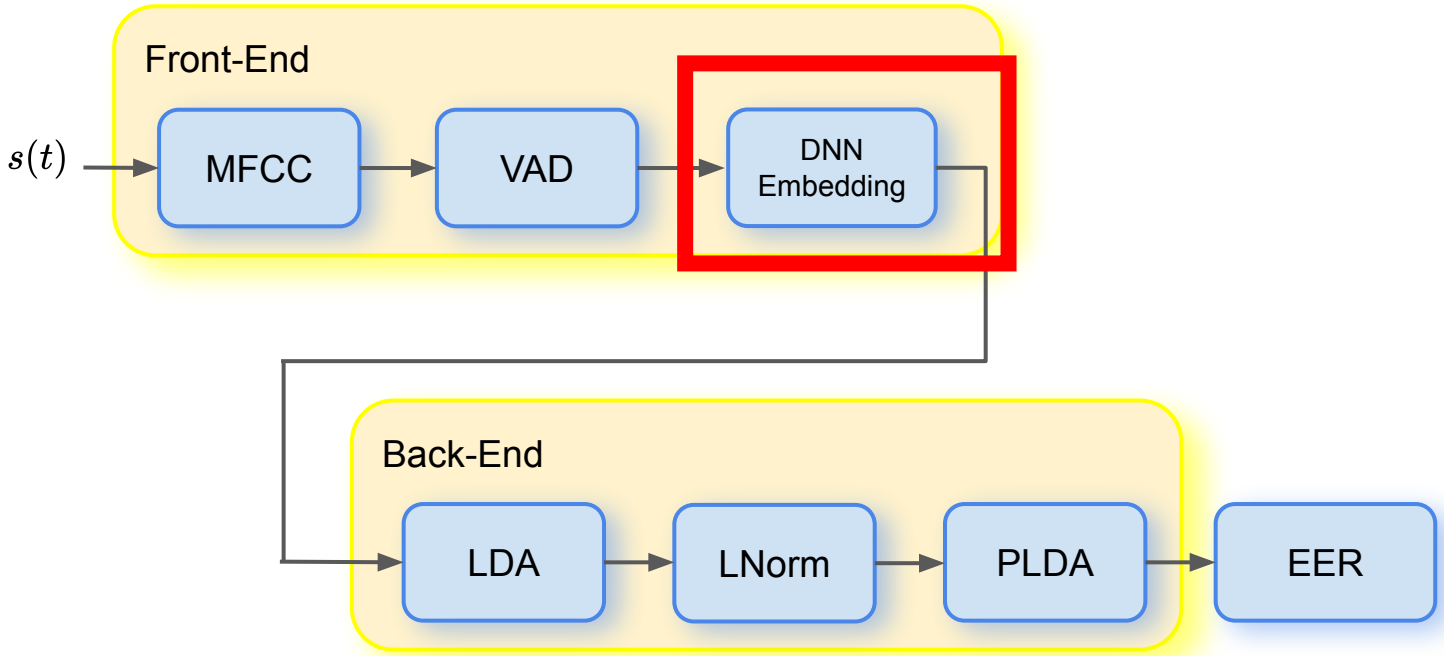
SITW: train x-vector network

- Step 011: run_011_train_xvector.sh
  - Optimize network by minimizing cross-entropy
  - We iterate for 2 epochs over the data.
  - Final model left in
    - exp/xvector_nnet_2a.1.voxceleb_small/final.raw
  - For the rest of experiments we will use pretrainied model with more data in
    - exp/xvector_nnet_2a.1.voxceleb_div2



SITW: extract x-vectors

- Step 030: run_030_extract_xvectors.sh
  - Extracts x-vectors
    - Voxceleb Training data
    - SITW Enrollment and test data
    - Combine SITW enrollment and test set into a unique list



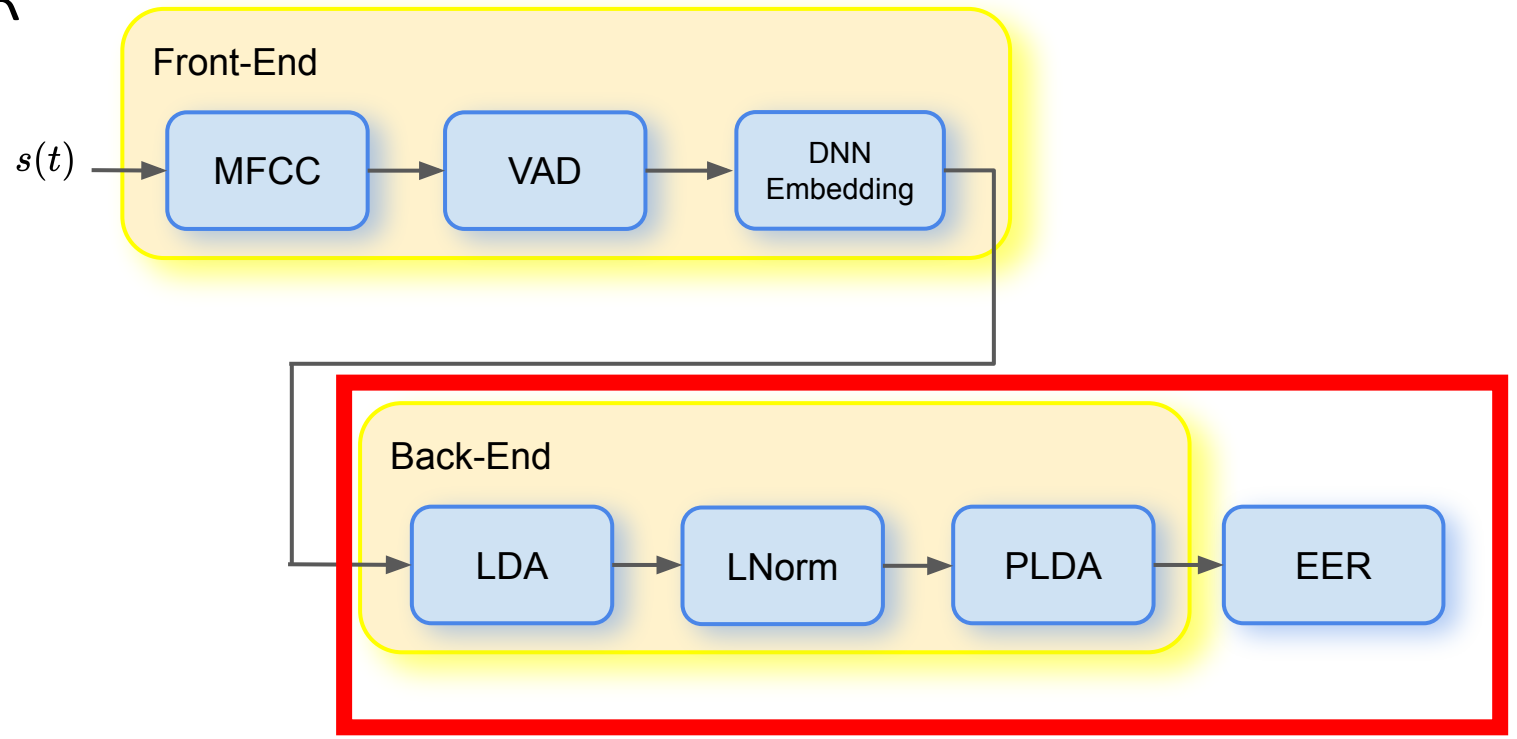
SITW: extract x-vectors

- Step 030: run_030_extract_xvectors.sh
  - Extracts x-vectors
    - Voxceleb Training data
    - SITW Enrollment and test data
    - Combine SITW enrollment and test set into a unique list
    - X-vectors are left in:
      - exp/xvectors/2a.1.voxceleb_div2/voxceleb_small/xvector.scp
      - exp/xvectors/2a.1.voxceleb_div2/sitw_dev_enroll/xvector.scp
      - exp/xvectors/2a.1.voxceleb_div2/sitw_dev_test/xvector.scp
      - exp/xvectors/2a.1.voxceleb_div2/sitw_combined/xvector.scp

```
for name in sitw_dev_enroll sitw_dev_test
do
    steps_kaldi_xvec/extract_xvectors.sh --cmd "$train_cmd --mem 6G" --nj 40 \
                                         $nnet_dir data/$name \
                                         $xvector_dir/$name
done
```

SITW: speaker detection back-end

- Step 040: run_040_eval_be.sh
  - Train LDA, Centering, Whitening and PLDA on Voxceleb
  - Evaluate SITW trial list
  - Calculate EER



SITW: back-end training

- Step 040: run_040_eval_be.sh
  - Back-end training
  - Call python script steps_be/train-be-v1.py
    - It gets:
      - Training list: data/voxceleb_small/utt2spk
      - Xvector file: exp/xvectors/2a.1.voxceleb_div2/voxceleb_small/xvector.scp
      - Output model dir: exp/be/2a.1.voxceleb_div2/lda200_splday150_v1_voxceleb_small/
      - LDA dimension, PLDA speaker space dimension, …

SITW: back-end training

- Step 040: run_040_eval_be.sh
  - Training steps:
    - Load training data
    - Train LDA
    - Apply LDA to training data
    - Train centering+whiteing
    - Apply centring+whitening to data
    - Train PLDA model
    - Save LDA+centering models into lda_lnorm.h5 file
    - Save PLDA model into plda.h5 file

```
from hyperion.transforms import TransformList, LDA, LNorm
from hyperion.helpers import PLDAFactory as F

# somehow read training data, e.g., use RandomAccessDatarReaderFactory
x, class_ids = read_train_data()

# train LDA
lda = LDA(lda_dim=lda_dim)
lda.fit(x, class_ids)

# apply LDA to training data
x_lda = lda.predict(x)

# train centering and whitening
lnorm = LNorm(name='lnorm’)
lnorm.fit(x_lda)

# apply centering, whitening and length norm.
x_ln = lnorm.predict(x_lda)

# create PLDA object
# The PLDA factory allows to select different PLDA types (frplda, splda, plda)
plda = F.create_plda(plda_type=‘splda’, y_dim=y_dim)
# train PLDA
elbo = plda.fit(x_ln, class_ids)

# save lda and length normalization as a list of transformations
preproc = TransformList(lda)
preproc.append(lnorm)
preproc.save('lda_lnorm.h5’)

# save plda model
plda.save('plda.h5')
```

SITW: back-end evaluation

- Step 040: run_040_eval_be.sh
  - Back-end evalutation
  - Call python script steps_be/eval-be-v1.py
    - It gets:
      - Enrollement list: data/sitw_dev_enroll/utt2spk
      - Trial list: data/sitw_dev_test/trials/core-core.lst
      - X-vector file: exp/xvectors/2a.1.voxceleb_div2/sitw_combined/xvector.scp
      - LDA, PLDA model
      - Output score file:
        - exp/scores/2a.1.voxceleb_div2/lda200_splday150_v1_voxceleb_small/plda/sitw_dev_core-core_scores

SITW: back-end evaluation

- Step 040: run_040_eval_be.sh
  - Eval steps:
    - Load x-vector preprocessing transforms:
      - LDA, centering+whitening.
    - Load evaluation data:
      - Trial file (Ndx class)
        - ndx.model_set
        - ndx.seg_set
      - X-vectors: x_enroll, x_test
    - Apply transformations to x-vectors
    - Compute LLR
    - Save scores (TrialScores class)
      - model_set, seg_set, scores

```
from hyperion.utils.trial_ndx import TrialNdx, TrialScores
from hyperion.helpers import PLDAFactory as F
from hyperion.transforms import TransformList

# load lda and length norm models
preproc = TransformList.load(preproc_file)

# load plda model
model = F.load_plda('splda', model_file)

# load ndx file and x-vectors
ndx, x_enroll, x_test = load_trial_data()


# appla LDA and lnorm to x-vectors
x_enroll = preproc.predict(x_enroll)
x_test = preproc.predict(x_test)

# compute PLDA llr scores
scores = model.llr_1vs1(x_enroll, x_test)

#save scores
s = TrialScores(ndx.model_set, ndx.seg_set, scores)
s.save_txt(score_file)
```

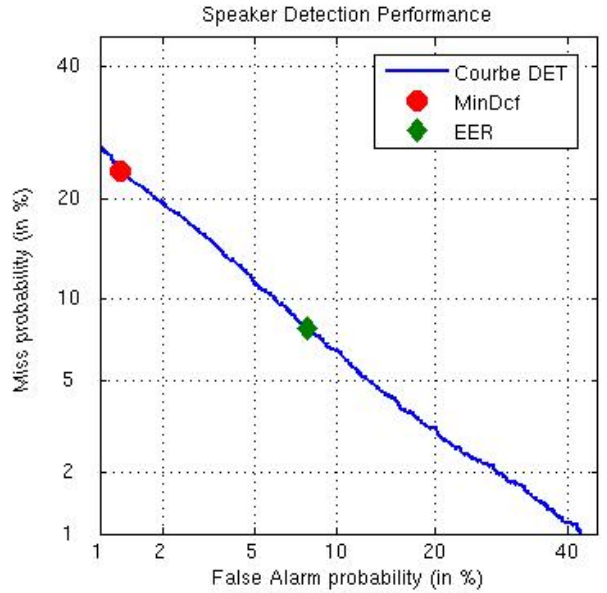
SITW: back-end evaluation

- Step 040: run_040_eval_be.sh
  - Output: score file
    - exp/scores/2a.1.voxceleb_div2/lda200_splday150_v1_voxceleb_small/plda/sitw_dev_core-core_scores

```
Score_file
Enrollment-id test-utterance-id log-likelihood-ratio-score
21417 aahtm -98.915693
21473 aahtm -72.799947
21520 aahtm 21.447149
21521 aahtm -36.370322
21689 aahtm -58.404719
21713 aahtm -7.025460
21752 aahtm -11.338147
21847 aahtm -46.686449
21898 aahtm -77.300953
```

SITW: calculate EER, DCF

- Step 040: run_040_eval_be.sh
  - Calculate
    - EER: $P_{FA}(\theta) = P_{Miss}(\theta)$
    - DCF: $C = P_{target}C_{Miss}P_{Miss}\,(\theta) + \left(1 - P_{target}\right)C_{FA}P_{FA}(\theta)$
  - local/score_dcf.py gets trial file and score file.




```
from hyperion.utils import TrialScores, TrialKey
from hyperion.metrics import fast_eval_dcf_eer as fast_eval

key = TrialKey.load_txt(key_file)  # read key
scr = TrialScores.load_txt(score_file) # read scores
tar, non = scr.get_tar_non(key). # separate scores into target and nontarget

priors = np.array([0.001, 0.005, 0.01, 0.05 ]) # target priors for DCF
# get EER min and actual DCF for all operating points
min_dcf, act_dcf, eer, _ = fast_eval(tar, non, priors)
```