SAMSUNG

Voice recognition

Before and after Deep Learning

Mikhail Kudinov

Samsung R&D Center Russia

March 29, 2019

Table of Contents

We are here!

This is the first slide

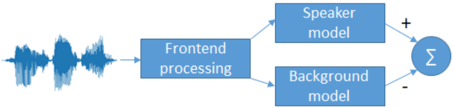
Ideas

See how this loooks

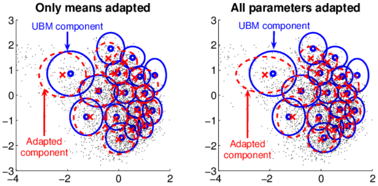
- Item 1
- Item 2

Universal Background Model

- Criterion:
  $\Lambda(X) = \log p(X|\theta_s) - \log p(X|\theta_{bck}) \geq C$
- Train separate model θ_{bck} called Universal Background Model
- All frames are assumed to be independent
- $p(X_t|\theta)$ is a GMM with diagonal covariance matrices Σ_i
- θ_{bck} is trained with EM algorithm on the whole collection of speech data
- Speaker model θ_s is an MAP update of θ_{bck}



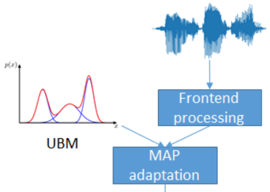
Likelihood ratio based system layout



MAP update of UBM

SVM on GMM supervectors

- Supervector m: do MAP-updates of means of the UBM and concatenate them
- Use lower-bound on KL-divergence as a distance measure:
  $d(m^a, m^b) = \frac{1}{2}\sum_i \pi_i (m_i^a - m_i^b)\Sigma_i^{-1}(m_i^a - m_i^b)$
- Use SVM on supervectors with the kernel defined above
- Nuisance attribute projection



$$m = \begin{pmatrix} m_1^1 \\ m_2^1 \\ \cdots \\ m_2^1 \\ \cdots \\ m_N^M \end{pmatrix}$$

Supervector computation

Extensions of supervector model

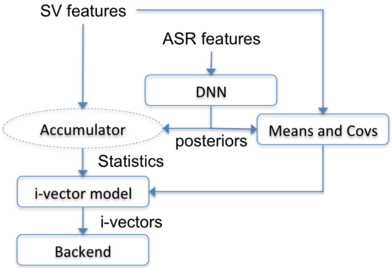
- SVM with simple cosine kernel on preprocessed supervectors
- Within-class Covariance Normalization $k(m_1, m_2) = m_1 W^{-1} m_2$, where W is a covariance matrix: $W = \frac{1}{S}\sum_{s=1}^{S}\frac{1}{n_s}\sum_{i=1}^{n_s}(m_i^s - \overline{m_s})(m_i^s - \overline{m_s})^t$
- SVM on LDA-projected supervectors
- Nuisance Attribute Projection $\arg\min_P \sum_{i,j} W_{i,j}\|Pm_1 - Pm_2\|_2^2$, where $W_{i,j} = 0$ iff $w1$ and $w2$ are from e.g. different microphones (nuisance channel direction)
- Joint Factor Analysis: $m = \mu + Vy + Ux + Dz$, where y is a speaker-dependent and x is a channel-dependent vector
- Probabilistic LDA (PLDA)

i-vectors

- Front-end Factor Analysis for a supervector M: $m = \mu + Tw$, where μ is the speaker- and channel-independent vector (UBM-vector)
- T is a rectangular matrix of low rank
- $w \sim \mathcal{N}(0, \mathbb{I})$ is called *i-vector*
- i-vector estimation for known matrix T for utterance u:
  - Use UBM Ω to get Viterbi or Baum-Welch estimations for GMM component c: $N_c = \sum_t P(c|y_t, \Omega)$; $F_c = \sum_t P(c|y_t, \Omega)(y_t - \mu_c)$
  - $w = (I + T^T \Sigma^{-1} N T)^{-1} T^T \Sigma^{-1} F$, where N and F are concatenations of N_c and F_c for all GMM components $\{c_i\}$
- T is estimated using ML algorithm

DNNs for better i-vector extraction

- Use DNN to obtain class-posterior for statistics collection *N* and *F* for i-vector calculation
- Classes are context-aware *senones*
- Requires ASR system trained separately
- Similarity score was computed on PLDA-projections



DNN/i-vector framework

```
OLTP test statistics:
    queries performed:
        read:                            716114
        write:                           255715
        other:                           102290
        total:                           1074119
    transactions:                        51139  (852.18 per sec.)
    deadlocks:                           12     (0.20 per sec.)
    read/write requests:                 971829 (16194.50 per sec.)
    other operations:                    102290 (1704.55 per sec.)

General statistics:
    total time:                          60.0098s
    total number of events:              51139
    total time taken by event execution: 479.6358
    response time:
         min:                                  2.03ms
         avg:                                  9.38ms
         max:                                 77.61ms
         approx.  95 percentile:              14.18ms

Threads fairness:
    events (avg/stddev):           6392.3750/15.62
    execution time (avg/stddev):   59.9545/0.00



OLTP test statistics:
    queries performed:
        read:                            747292
        write:                           266853
        other:                           106745
        total:                           1120890
    transactions:                        53367  (889.03 per sec.)
    deadlocks:                           11     (0.18 per sec.)
    read/write requests:                 1014145 (16894.47 per sec.)
    other operations:                    106745 (1778.25 per sec.)

General statistics:
    total time:                          60.0282s
    total number of events:              53367
    total time taken by event execution: 479.6704
    response time:
         min:                                  1.93ms
         avg:                                  8.99ms
         max:                                101.82ms
         approx.  95 percentile:              13.52ms

Threads fairness:
    events (avg/stddev):           6670.8750/11.55
    execution time (avg/stddev):   59.9588/0.01

~
~
NORMAL  withperf.txt                                   text  utf-8  146 words  1%  1/52 :  1
"withperf.txt" 52L, 2053C
```

This slide has code blocks

```python
import numpy as np

def incmatrix(genl1,genl2):
    m = len(genl1)
    n = len(genl2)
    M = None #to become the incidence matrix
    VT = np.zeros((n*m,1), int)  #dummy variable
```

We are here!

This is the second slide

A bit more information about this

Alert block

Alert text

We are here!

Generic slide

A bit more information about this

We are here!

Generic slide

A bit more information about this

We are here!

Generic slide

A bit more information about this

We are here!

Generic slide

A bit more information about this